VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Porovnávání dat mezi relační databází a adresářovou službou

Comparing data between relational database and directory service

2009 David Vaštík

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 1. května 2009 .....

Rád bych na tomto místě poděkoval mému zadavateli, panu Ing. Martinu Lasoňovi, za užitečné upozornění na chyby a připomínky, které vedly ke zdokonalení mé práce.

Abstrakt

Hlavním cílem mé diplomové práce bylo vytvoření programu pro porovnávání dat uložených v relační databázi a k nim korespondujícím datům v adresářové službě. Program by měl nalézt nekorespondující data a vypsat v čem se tato data liší. Součástí mé práce bylo prostudování témat relačních databází, adresářových služeb X.500 a principy fungování identity managementu a také, jak je používán na VŠB – Technické univerzitě Ostrava.

Klíčová slova: relační databáze, adresářová služba, LDAP, správa identit, porovnávání

Abstract

The main goal of my thesis was to create the program for comparing data stored in relational database and corresponding data in directory service. The program had to find a noncorresponding data and list their differences. Part of my work was to study topics of relational databases, directory services X.500 and principles of identity management also as used in the VŠB – Technical University of Ostrava.

Keywords: relational database, directory service, LDAP, identity management, comparing

Seznam použitých zkratek a symbolů

API	–	*Application Programming Interface* Rozhraní pro programování aplikací. Jde o sadu tříd a rozhraní nějaké knihovny, které může využívat programátor.
ACL	–	*Access Control List* Seznam pro řízení přístupu spojený s nějakým objektem. Definuje kdo nebo co má jaké oprávnění k provádění operací nad tímto objektem.
DFD	–	*Data Flow Diagram* Diagram datových toků. Ve fázi analýzy navrhovaného systému slouží k popisu jeho funkcí.
DIT	–	*Directory Information Tree* Hierarchická struktura pro organizaci objektů v adresáři.
DSML	–	*Directory Service Markup Language* Nástroj pro reprezentaci dat z adresářového serveru ve formátu XML.
IDM	–	*Identity management* Nástroj pro centrální správu identit v počítačové síti.
JDBC	–	*Java Database Connectivity* Definuje jednotné rozhraní pro přístup k relačním databázím v programovacím jazyce Java.
JNDI	–	*Java Naming and Directory Interface* Definuje jednotné rozhraní pro přístup k jmenným a adresářovým službám v programovacím jazyce Java.
LDAP	–	*Lightweight Directory Access Protocol* Standardní protokol pro ukládání a přístup k datům na adresářovém serveru.
LDIF	–	*LDAP Data Interchange Format* Standardizovaný textový formát pro výměnu dat v adresářových službách.
OID	–	*Object Identifier* Jedinečný globální identifikátor sloužící k identifikaci objektů v adresářových službách
SASL	–	*Simple Authentication and Security Layer* Obecná metoda pro zlepšení ověřování identity uživatele přistupujícího na server.

SSL – *Secure Socket Layer*
Protokol pro bezpečný přenos dat v internetu. Využívá šifrovaného přenosu a autentizace uživatelů pomocí digitálních certifikátů.

SQL – *Structured Query Language*
Standardizovaný dotazovací jazyk používaný pro práci s daty v relačních databázích

TLS – *Transport Layer Security*
Následník protokolu SSL pro bezpečný přenos dat v internetu.

UML – *Unified Modeling Language*
Grafický jazyk využívaný v softwarovém inženýrství pro vizualizaci, specifikaci, navrhování a dokumentaci programových systémů.

URI – *Uniform Resource Identifier*
Řetězec znaků s definovanou strukturou, sloužící k přesné specifikaci umístění zdrojů informací na internetu. Definuje doménovou adresu serveru, umístění zdroje na serveru a protokol, kterým je možné zdroj zpřístupnit.

XML – *eXtensible Markup Language*
Značkovací jazyk určený především pro výměnu různých druhů dat mezi aplikacemi a pro publikování dokumentů.

Obsah

Seznam tabulek

Seznam obrázků

1 Úvod

Diplomová práce si klade za cíl zejména navrhnout a implementovat program pro porovnávání dat mezi relační databází a adresářovou službou. Práce se však nezabývá pouze popisem vývoje aplikace, ale také tématy s touto aplikací spojenými, tedy relačními databázemi, adresářovými službami a správou identit. Celá práce je rozdělena do čtyř tematických kapitol.

Práce začíná kapitolou o relačních databázích. Stručně je charakterizuje, zejména však vysvětluje způsob reprezentace dat v těchto databázích. Kapitola zahrnuje také popis triggerů, které jsou nedílnou součástí většiny relačních databází. Poukazuje na to, k čemu nám triggery slouží, jaké druhy existují a kdy jsou používány. Zároveň popisuje syntaxi jejich vytváření v databázích DB2.

Ve druhé kapitole jsou popsány adresářové služby. V úvodu jsou nastíněny rozdíly oproti relačním databázím, dále jsou zde popsány protokoly, pomocí kterých lze přistupovat k adresářovým službám. Nakonec je uvedeno několik konkrétních implementací LDAP adresáře.

Třetí kapitola pojednává o moderním způsobu řízení uživatelských účtů v organizaci pomocí správy identit. Popisuje nesporné výhody zavedení jednotné správy uživatelů v rozrůstajících se organizacích s více heterogenními systémy. Rozebírá také nejznámější produkty vyskytující se na aktuálním trhu a poukazuje na to, jak je správa identit implementována na VŠB – Technické univerzitě Ostrava.

Poslední kapitola se poté zabývá samotným vývojem aplikace. Detailněji seznamuje čtenáře se specifikací požadavků a následnou funkční a datovou analýzou. V části popisu implementace jsou také rozebrány technologie, které byly při vývoji použity.

2 Relační databáze

Spousta lidí chápe pojem relace jako vztah mezi tabulkami a domnívají se, že odtud vznikl pojem relační databáze, což však není pravda. Pojem je odvozen od relačního modelu, na kterém jsou tyto databáze založeny [1]. S tímto modelem přišel v roce 1970 pracovník firmy IBM, Dr. E. F. Codd. Je založen na matematické teorii a predikátové logice a definuje způsob reprezentace dat v databázi, možné operace nad nimi či způsob jejich ochrany. Veškeré operace nad daty jsou definovány pomocí základních operací, jako jsou sjednocení, průnik, rozdíl či kartézský součin. Relační databáze je založena na tabulkách, kde sloupce odpovídají jednotlivým atributům a řádky postupně samotným záznamům. Pojmem **relace** tak rozumíme tabulku databáze, která je definována jako podmnožina kartézského součinu všech možných hodnot z jednotlivých sloupců.

Každý sloupec má definován řadu vlastností, zejména pak datový typ hodnot v něm uložených. Hodnoty některých sloupců mají zvláštní význam a hrají velkou roli v relačních databázích. Jedná se o hodnoty nazvané primární klíč a cizí klíč.

- **Primární klíč** je jedinečný identifikátor každého řádku. V každé relaci tedy může existovat vždy pouze jeden záznam s daným primárním klíčem. Klíč však nemusí odpovídat pouze jediné hodnotě, ale může být určen složením hodnot z více atributů. Vždy však musí být zaručena ona jedinečnost záznamu. Hodnoty těchto atributů tak musí být vždy bezpodmínečně zadány pro každý záznam v relaci. Často jsou tyto hodnoty automaticky přiřazovány systémem při vkládání nového záznamu. Obvykle se jedná o automatické číslování těchto záznamů. Všechna data v databázi jsou přístupná pomocí kombinace jména tabulky, hodnoty primárního klíče a názvu sloupce.

- **Cizí klíč**, někdy také nazývaný nevlastní, hraje hlavní roli při definování vztahů mezi tabulkami. Hodnoty primárního klíče ve zdrojové tabulce a k nim odpovídající hodnoty cizího klíče v „cizí“ tabulce určují, které záznamy jsou vzájemně propojeny. Cizí klíče jsou úzce spjaty s referenční integritou dat, kterou se budu zabývat dále.

Atributy s těmito vzájemně propojenými klíči musí být vždy shodného typu.

Integritní omezení

Integritní omezení definují pravidla pro zadávání nových hodnot. Hodnota tak musí například odpovídat zadanému datovému typu či formátu nebo třeba udává, zda může být atribut nevyplněn. Jedním z důležitých omezení je takzvaná **referenční integrita dat**, nástroj, který pomáhá udržovat vztahy mezi záznamy v propojených tabulkách. Definuje se také, co má být v případě porušení integrity dat provedeno. To je nazýváno aktivní referenční integritou.

V případě přidávání nových záznamů do podřízené tabulky nebo při úpravě stávajících, systém kontroluje, zda v nadřízené tabulce existuje primární klíč s hodnotou zadaného cizího klíče.

V případě odebírání záznamu nebo úpravě hodnoty primárního klíče v nadřízené tabulce se v podřízené tabulce kontroluje, zda k němu existují odpovídající záznamy. Pokud ano, systém může nahlásit chybu nebo provést akci, která byla pro nastalou událost definována. Různé akce lze definovat jak pro případy odstranění záznamu z nadřízené tabulky, tak při jeho editaci. Možné jsou čtyři následující akce:

- `SET NULL` – v záznamech podřízené tabulky s odpovídajícím klíčem se hodnoty cizího klíče nastaví na NULL
- `SET DEFAULT` – v záznamech podřízené tabulky s odpovídajícím klíčem se hodnoty cizího klíče nastaví na jejich výchozí hodnotu zadanou v definici tabulky
- `CASCADE` – v záznamech podřízené tabulky s odpovídajícím klíčem se hodnoty cizího klíče nastaví na hodnotu odpovídajícího primárního klíče z nadřízené tabulky. V případě odebrání záznamu z nadřízené tabulky jsou zrušeny i záznamy z podřízené tabulky.
- `NO ACTION` – neprovede se žádná akce (je vypsána chyba porušení integrity dat)

Vztahy mezi tabulkami

Tabulky mohou v databázi existovat samostatně, tedy bez vztahů k ostatním tabulkám. Existuje-li však mezi nimi závislost, pak definujeme tři typy vztahu:

- **1:1** – tento vztah používáme v případech, kdy záznamu v jedné tabulce odpovídá maximálně jeden záznam v tabulce s ní spojené. Představme si například tabulky `Zamestnanec` a `Katedra`. Záznamy v těchto tabulkách budou ve vztahu 1:1, protože každá katedra má nejvýše jednoho vedoucího a zároveň každý zaměstnanec může být vedoucím pouze na jediné katedře.
- **1:N** – u tohoto vztahu je se záznamem svázáno více záznamů v podřízené tabulce. Každý záznam v podřízené tabulce může být spjat vždy pouze s jedním záznamem z nadřízené tabulky. Jedná se o nejvyužívanější druh vztahu, protože odpovídá mnoha situacím z reálného života. Příkladem mohou být tabulky `Uzivatel` a `Mail`, tedy každý uživatel může mít několik e-mailů, ale každý e-mail může patřit pouze jedinému uživateli.
- **M:N** – umožňuje několika záznamům z jedné tabulky přiřadit několik záznamů z tabulky druhé. V praxi se tento vztah rozkládá na dva vztahy typu 1:N a 1:M a jejich záznamy jsou propojeny s doplňující spojovací tabulkou. Ve vztahu M:N mohou být třeba záznamy z tabulek `Uzivatel` a `Skupina`, kdy je zřejmé, že skupiny mohou obsahovat několik uživatelů a zároveň uživatel může být členem více skupin. Po rozložení by vznikla nová tabulka, určující, který uživatel patří do které skupiny.

Pojmem relační databáze však nemusíme chápat pouze jako samotné úložiště dat. Může s ním být spjat také systém, který s databází pracuje. Ten je nazýván systémem řízení báze dat (SŘBD).

Jazykem pro ovládání databáze je v současné době obvykle strukturovaný dotazovací jazyk SQL. V další části textu bude tohoto jazyka často využíváno, jeho popis je však nad rámec této práce a nebudu se jím tedy dále zabývat.

2.1 Triggery

V následující části se zmíním o triggerech [2]. Jedná se o databázové objekty, do češtiny překládané jako spouštěče, které jsou aktivovány při nějaké události na tabulce v databázi. A to při přidání nového záznamu nebo při editaci či odebrání již existujícího. Triggery nám můžou výrazně usnadnit práci zejména při tvorbě databázových aplikací. O kus práce prováděné při zmiňovaných událostech se stará trigger na serveru a nemusí se tak implementovat v aplikaci nebo ještě lépe v žádné z aplikací k databázi přistupujících. Také při případné změně logiky postačí změnit pouze definici triggeru v databázi bez nutnosti jakéhokoliv zásahu do těchto aplikací. Jejich problematika se však v různých databázových systémech odlišuje. Jedná se jak o rozdíly v syntaxi, tak o rozdíly v sémantice. Budu se tedy konkrétně věnovat databázi DB2, která je používána při spravování identit na VŠB. V tomto systému je problematika triggerů zaměřena na jednoduchost syntaxe a jednoznačnost sémantiky.

Vytvoření triggeru

Při vytváření triggerů si musíme promyslet jaké akce, na které tabulce, při jaké události či v jaké chvíli se budou provádět. Tímto se budu v této části věnovat. Níže je potom uvedena kompletní syntaxe pro vytvoření triggeru v databázi DB2 pomocí SQL a několik ukázek různých triggerů.

Definice triggeru začíná klíčovými slovy `CREATE TRIGGER` a jeho názvem, který musí být jedinečný v rámci celé databáze. Podle toho, kdy jsou triggery spouštěny, se rozdělují na tři základní kategorie:

- **before triggery** – jsou vyvolány ještě před provedením definované události. Tyto triggery mohou například kontrolovat hodnoty, které mají být vloženy nebo upraveny v databázi ještě předtím, než se databáze opravdu aktualizuje. Jsou tak schopny rozšířit kontrolu pomocí integritních omezení. Jsou schopny kontrolovat, zda nově ukládaná hodnota splňuje podmínky pro přidání nebo úspěšnou změnu. Například víme-li, že zaměstnanci může být navýšen plat maximálně o 10%, trigger při nesplnění této podmínky nepovolí editaci této hodnoty. Nemohou však měnit databázi pomocí příkazů `INSERT`, `UPDATE` nebo `DELETE`. V syntaxi je tento druh triggeru uveden klíčovým slovem `BEFORE`. U všech before triggeru je povinné zároveň klíčové slovo `NO CASCADE` zamezující vyvolání dalších triggerů.
- **after triggery** – jsou vyvolány po provedení definované události a slouží tak k implementaci aplikační logiky. Například po vložení nového záznamu do tabulky může provést blok příkazů, aktualizujících další tabulky. V syntaxi jsou označovány klíčovým slovem `AFTER`.

- **instead of triggery** – jsou vyvolány namísto definované události, kterou trigger hlídá. Tyto triggery nejsou spojeny se samotnými tabulkami, nýbrž s pohledy. V syntaxi jsou označovány klíčovým slovem `INSTEAD OF`.

Triggery lze definovat pro tři druhy událostí, při kterých bude trigger vyvolán:

- `INSERT` – přidání nového záznamu
- `UPDATE [OF COLUMN <seznam_sloupců>]` – upravení existujícího záznamu. Trigger může být aktivován také pouze v případě, že je upravován pouze některý ze zadaných atributů (sloupců).
- `DELETE` – odebrání záznamu

Každý vytvářený trigger může být definován pouze na jednu z těchto událostí. Jednu událost však může hlídat více triggerů. Pořadí vykonání u více triggerů najednou nelze určit v syntaxi triggeru a je v takovém případě dáno časem vytvoření triggerů. Pokud hlídaná událost ovlivňuje více než jeden záznam, pak se může spouštět buďto pro každý ovlivněný záznam zvlášť, nebo pouze jednou při vyvolání. Například u příkazu `INSERT INTO tab1 SELECT * FROM tab2;` může být trigger hlídaný událost `INSERT` na tabulce `tab1` vyvolán při každém pokusu o vložení záznamu z tabulky `tab2` do tabulky `tab1` nebo pouze jednou pro všechny záznamy společně. Podle toho triggery rozdělujeme na dva druhy:

- **řádkové triggery** – vyvolávány pro každý záznam zvlášť. V syntaxi jsou označovány klíčovým slovem `FOR EACH ROW`. Aktualizace databáze v tomto případě proběhne až po spuštění triggerů pro všechny záznamy.
- **příkazové triggery** – vyvolávány pouze jednou pro celý příkaz. Jsou označovány klíčovým slovem `FOR EACH STATEMENT`

Before a instead of triggery jsou vždy řádkové a nikdy příkazové. After triggery mohou být jak řádkové, tak příkazové. V případě, že událost neovlivňuje žádný záznam, pak se řádkový trigger nespustí ani jednou, ale příkazový trigger se i v tomto případě jednou spustí.

Podle události, se kterou je trigger spjat si může vytvořit reference na záznamy nebo tabulky, se kterými může pracovat. Definují se u klíčového slova `REFERENCING` a pro řádkové triggery si pomocí `OLD AS` a `NEW AS` může vytvořit reference na záznamy nebo pro příkazové triggery pomocí `OLD_TABLE AS` a `NEW_TABLE AS` reference na tabulky. Pomocí těchto referencí poté můžeme třeba porovnat hodnoty uložené v databázi s hodnotami, na které je chceme změnit. Pochopitelně insert triggery mohou využívat referencí pouze na nové záznamy, které mají být přidány do tabulky, delete triggery pouze na záznamy, které mají být odebrány a update triggery mohou mít reference jak na záznamy uložené v databázi, tak na záznamy, kterými mají byt nahrazeny.

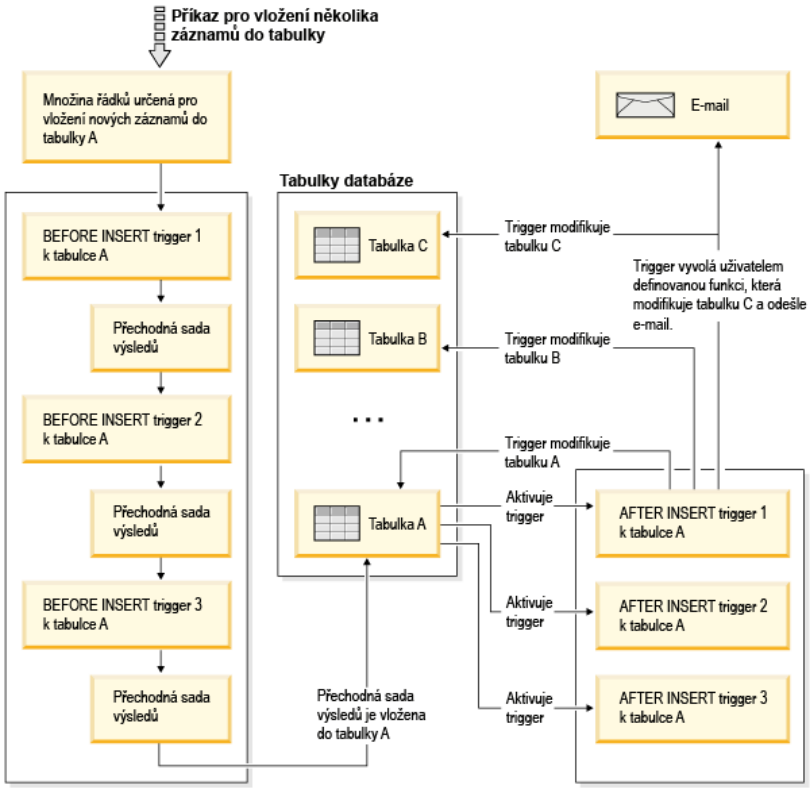
Dále je nutno definovat název tabulky, ke které je vytvářený trigger spjat a může obsahovat doplňující podmínku, při jejíž nesplnění nebude blok příkazů triggeru spuštěn.

U instead of triggerů nelze tuto podmínku využít.

V bloku prováděných příkazů vymezených klíčovými slovy `BEGIN ATOMIC` a `END` se mohou vyskytovat jak složitější databázové příkazy, tak nedatabázové operace, jako odeslání e-mailu či zápis dat do souboru v souborovém systému.

```
CREATE TRIGGER <název_triggeru>
[NO CASCADE BEFORE | AFTER | INSTEAD OF]
{INSERT | DELETE | UPDATE [OF COLUMN <seznam_sloupců>]}
ON <název_tabulky>
[FOR EACH ROW MODE DB2SQL | FOR EACH STATEMENT ]
[REFERENCING
   [OLD AS <označení_původních_hodnot>]
   [NEW AS <označení_nových_hodnot>]
   [OLD_TABLE AS <označení_původní_tabulky>]
   [NEW_TABLE AS <označení_nové_tabulky>]]
WHEN <podmínka>
BEGIN ATOMIC
   <blok_příkazů>
END
```

Výpis 1: Syntaxe pro definici triggeru v DB2



Obrázek 1: Průběh spouštění triggerů

Na obrázku 1 [2] na straně 9 je znázorněn průběh spouštění několika jak before, tak after triggerů spjatých k zadané tabulce při akci vložení nových záznamů. Přirozeně jsou nejdříve postupně spouštěny before triggery. Výstup jednoho triggeru je vždy vstupem toho následujícího. Výstup posledního before triggeru může být vložen do tabulky a následně mohou být vyvolány after triggery.

Příklady vytvoření triggerů

V následující části jsou popsány ukázky několika triggerů, které byly nebo jsou používány na VŠB – TUO.

Prvním příkladem je before update trigger, který při pokusu o změnu expirace uživatelského účtu způsobí, že k nové hodnotě bude přičteno dvacet hodin a změní tak konec platnosti účtu z půlnoci na osm hodin večer. Změna však proběhne pouze v případě, že nová hodnota `loginexpirationtime` není nulová. Vidíme, že pro účely porovnávání hodnot jsou vytvořeny reference `old` na starý a `new` na nový záznam.

```
CREATE TRIGGER INDIRECT.TRG_USR_B_U
NO CASCADE BEFORE UPDATE
ON INDIRECT.USR
REFERENCING
   NEW AS new
   OLD AS old
FOR EACH ROW MODE DB2SQL
WHEN (new.loginexpirationtime IS NOT NULL
   AND new.loginexpirationtime <> old.loginexpirationtime)
BEGIN ATOMIC
   IF  new.loginexpirationtime = TIMESTAMP_ISO('9999-12-31') THEN
      SET new.loginexpirationtime = TIMESTAMP_ISO('2030-12-31');
   END IF;
   SET new.loginexpirationtime = new.loginexpirationtime + 20 HOURS;
END;
```

Výpis 2: Ukázka vytvoření before triggeru pro událost UPDATE

Dalším příkladem je after insert trigger svázaný s tabulkou `app_role_type` ve schématu `role`. Ten se po vložení záznamů do této tabulky pokusí vložit nové záznamy do tabulky `grp` ve schématu `indirect` a vytvořit tak k novým aplikačním rolím příslušné LDAP skupiny, pokud takové ještě neexistují. V příkladu vidíme, že v bloku příkazů lze využít i volání na uloženou proceduru, která v našem případě sestavuje jméno pro novou skupinu.

```
CREATE TRIGGER ROLE.TRIG_APP_ROLE_T_I
AFTER INSERT
ON ROLE.APP_ROLE_TYPE
REFERENCING NEW AS new
FOR EACH ROW MODE DB2SQL
WHEN ( ldap_name IS NOT NULL)
BEGIN ATOMIC
```

```
    DECLARE role_name, ou VARCHAR(64);
    CALL role.create_name (new.id_is_component, new.ldap_name, ou, role_name);
    IF NOT EXISTS (SELECT idg FROM indirect.grp WHERE cn = role_name AND id_group_type =
        8) THEN
      INSERT INTO indirect.grp (cn, o, ou, id_group_type, id_app_role_type) VALUES (UPPER(
          role_name), 'APPS', UPPER(ou), 8, new.id_app_role_type);
    END IF;
END;
```

Výpis 3: Ukázka vytvoření after triggeru pro událost INSERT

Posledním příkladem je after delete trigger svázaný s tabulkou `tuocard` ve schématu `hr`. Ten se po odebrání záznamů pokusí nalézt jim odpovídající záznamy v tabulce `usr_card` ve schématu `indirect` a pokud existují, také je odebere.

```
CREATE TRIGGER HR.TRIG_TUOCARD_D
AFTER DELETE
ON HR.TUOCARD
REFERENCING OLD AS old
FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
    DECLARE uid INTEGER;
    SET uid = (SELECT idu FROM indirect.usr WHERE id_person = old.id_person);
    IF uid IS NOT NULL THEN
      DELETE FROM indirect.usr_card WHERE idu = uid AND tuocardmd5 = old.chip_md5_number;
    END IF;
END;
```

Výpis 4: Ukázka vytvoření after triggeru pro událost DELETE

3 Adresářové služby

Adresář

Adresářem rozumíme speciální databázi, určenou pro uchovávání velkého množství dat různého charakteru. Slouží nám k organizaci a sdružování dat do skupin, aby se v nich uživatel lépe orientoval. Adresáře jsou na rozdíl od relačních databází optimalizovány pro velmi časté čtení a s ním spojené vyhledávání, kdy tisícovky požadavků na vyhledávání odpovídají jedinému požadavku na aktualizaci dat v adresáři. Nevýhodou oproti relačním databázím je to, že adresáře nepodporují složité transakce a kontrolu referenční integrity. Adresářem může být chápán například organizovaný seznam kontaktů nebo seznam uživatelských kont v počítačových systémech. Jedná se vlastně o hierarchický seznam objektů a atributů těchto objektů, porovnatelný se strukturou adresářů a podadresářů souborového systému.

Adresářová služba

Adresářová služba je specializovaná aplikace, která přistupuje k adresáři a umožňuje práci s ním. Umožňuje tak ukládat a organizovat informace v adresáři. Adresářová služba je mnohdy považována za shodný pojem jako adresář, což však není žádoucí, protože s pojmem adresářová služba jsou složeny následující částí:

- informace v adresáři
- serverový software poskytující uložené informace
- klientský software zabývající se zprostředkováním informací pro uživatele
- hardware pro serverový a klientský software
- podpůrný software jako jsou operační systémy a ovladače zařízení
- síťová infrastruktura, která je schopna spojit klienty se servery
- bezpečnostní politika, která specifikuje oprávnění přístupu k záznamům, aktualizaci dat, apod.
- procedury, které adresářové služby používají pro údržbu a monitorování
- software používaný pro údržbu a monitorování adresářových služeb

X.500

V době velkého rozvoje internetu a aplikací, využívajících síťové prostředky, vznikala také spousta specializovaných adresářů. Postupem času tak byl navržen standard určující pravidla pro adresářové služby nazvaný X.500. Byl vyvinut organizací CCITT (Consultative Committee on International Telephony and Telegraphy) a je postaven na rodině protokolů ISO/OSI. Snažil se pokrýt všechny existující varianty adresářových služeb, jeho implementace je tedy dosti složitá a aplikace tak většinou implementovaly pouze některé jeho

části. To však opět vedlo k tomu, že někdy nebyly schopny mezi sebou komunikovat, i když využívaly shodného standardu. V praxi se tak pro svou velikost a složitost tento model ne zcela prosadil.

Lightweight Directory Access Protocol (LDAP)

LDAP je protokol pro přístup k adresářovým službám [3]. Jak již název odpovídá, jedná se o odlehčenou verzi protokolu DAP, který byl odvozen od standardu X.500 a byl využíván ke komunikaci mezi klientem a adresářovým serverem. Velký rozdíl oproti protokolu DAP je ten, že LDAP pracuje nad protokoly TCP/IP. Některé operace zjednodušuje a některé dokonce vynechává. Jedná se o jeden z nejpodporovanějších a nejčastěji používaných adresářových protokolů.

3.1 LDAP adresář

Časem po vývoji protokolu vznikl také samotný LDAP adresář, jehož služby jsou samozřejmě poskytovány prostřednictvím protokolu LDAP. Jeho popis lze rozdělit do čtyř modelů. (viz. [4, Adresářové služby])

3.1.1 Informační model

Úkolem informačního modelu LDAP je definovat datové typy a informace, které lze v adresářovém serveru ukládat. Informační model je založen na záznamech, které obsahují informace o nějakém konkrétním objektu, jako je uživatel či počítač. Pod pojmem záznam si můžeme představit souhrn atributů, a k nim příslušné hodnoty. Atributy nesou informaci o stavu daného záznamu. Každý takový atribut může obsahovat žádnou, jednu nebo i více hodnot. Prakticky je vícehodnotový atribut realizován tak, že je použit vícekrát s různými hodnotami. Všechny tyto hodnoty však musí odpovídat datovému typu atributu. Výčet typů je uveden v tabulce 1.

Datový typ	**Popis**
int	integer
bin	binární data
cis	(case ignore string) řetězec bez ohledu na velikost písmen při porovnávání
ces	(case exact string) řetězec s ohledem na velikost písmen při porovnávání
dn	(distinguished name) jedinečné rozlišující jméno v rámci stromové struktury
tel	telefonní číslo při porovnávání jsou ignorovány mezery a pomlčky, není brán ohled na velikost písmen

Tabulka 1: Datové typy atributů

Objektové třídy

Objektové třídy definují soubor atributů, pomocí kterých lze popsat nějaký konkrétní objekt, například uživatele systému. Tyto třídy bývají často odvozeny od nadřazených tříd tak, že jsou doplňovány o další potřebné atributy. Každý záznam přidávaný do adresáře musí být instancí některé z definovaných tříd.

- `ABSTRACT` - Abstraktní třída, od které se odvozují nové třídy. Samy nemohou být vzorem pro nové instance záznamů. Typickým příkladem je třída `top`, od které jsou odvozeny další třídy.
- `STRUCTURAL` - Každý záznam musí ve své definici obsahovat odkaz alespoň na jednu třídu tohoto typu. Většina hlavních atributů je definována právě ve třídách tohoto typu. V případě odvolání záznamu na více tříd, musí být tyto třídy v dědičném vztahu.
- `AUXILIARY` - Třída, která je doplňkovou třídou k ostatním. Velmi často má jako svého předka třídu `top`. Lze ji přiřadit ke každému typu záznamu, v záznamu je však třeba použít minimálně jednu třídu typu `STRUCTURAL`, jak již bylo zmíněno.

Objektová třída také definuje, které atributy musí, a které nemusí být při vytváření nových instancí vyplněny.

- `MUST` - atributy uvedené u tohoto klíčového slova musí být nutně vyplněny, aby mohl být záznam přidán do adresáře, případně upraven
- `MAY` - atributy uvedené u tohoto klíčového slova můžou, ale nemusí být vyplněny

```
objectclass ( 2.5.6.6
  NAME 'person'
  DESC 'RFC2256: a person'
  SUP top
  STRUCTURAL
  MUST ( sn $ cn )
  MAY ( userPassword $ telephoneNumber $
    seeAlso $ description )
)
```

Na příkladu vidíme definici objektové třídy nazvané 'person'. Před názvem třídy vidíme ještě globální identifikátor, tzv. OID (Object Identifier), pomocí kterého jsou identifikovány všechny objekty. Tento identifikátor je jedinečný v rámci celého světa a nejde jej tedy volit libovolně. Každé organizaci je přiděleno jedno výchozí OID a objektům jsou poté identifikátory přidělovány hierarchicky pomocí suffixů pro všechny objektové třídy, atributy atd. Dále následuje popis třídy u klíčového slova `DESC`, název třídy, ze které dědíme u klíčového slova `SUP` a klíčové slovo `STRUCTURAL` definující typ nově vytvářené třídy. U již zmiňovaných klíčových slov `MUST` a `MAY` jsou uvedeny seznamy atributů, které musí a nemusí být vyplněny při vytváření nových instancí.

Schéma

Pod pojmem schéma si představme soubor povolených objektových tříd a k nim náležících atributů. Jinými slovy definuje strukturu možný obsah každého objektu, který bude v adresáři vytvořen. Součástí adresářových serverů jsou standardizovaná schémata, která je možno využít.

Atribut, alias	Syntaxe	Popis
commonName, cn	cis	veřejné jméno položky
givenName	cis	křestní jméno
sureName, sn	cis	příjmení
uid	cis	id uživatele
telephoneNumber	tel	telefonní číslo
description	cis	popis
mail	cis	adresa elektronické pošty
jpegPhoto	bin	portrét osoby, kterou položka reprezentuje
organizationalUnitName, ou	cis	jméno organizační jednotky
organization, o	cis	jméno organizace

Tabulka 2: Ukázka několika běžně používaných atributů

3.1.2 Jmenný model

Položky jsou organizovány do hierarchické struktury zvané Directory Information Tree (DIT). Pro jednoznačnou identifikaci objektů v rámci této struktury je používáno rozlišovacího jména DN (distinguished name). Toto jméno v podstatě určuje polohu objektu v rámci stromu a je popsáno seznamem uzlů od objektu až ke kořeni stromu oddělených čárkami. Např. `dn: cn=vas186,o=6`

V rámci jedné úrovně ve větvi lze objekt identifikovat také pomocí relativního význačného jména RDN. Toto jméno je definováno pouze jako název atributu a k němu odpovídající hodnota a proto bychom v tomto případě neměli volit atributy, které můžou v rámci této úrovně obsahovat více shodných hodnot. Např. `givenName=David`. Vhodnější tak bude například atribut `uidNumber=10062` nebo `cn=vas186`.

3.1.3 Funkční model

Funkční model definuje, co se může provádět s informacemi v adresáři. Jedná se o devět operací, které jsou zařazeny do tří funkčních oblastí.

Autentizace

bind

Navázání spojení mezi LDAP serverem a klientem. Při navázání spojení probíhá autentizace. Možnosti autentizace jsou popsány v bezpečnostním modelu.

unbind

Ukončení spojení mezi LDAP serverem a klientem.

abandon

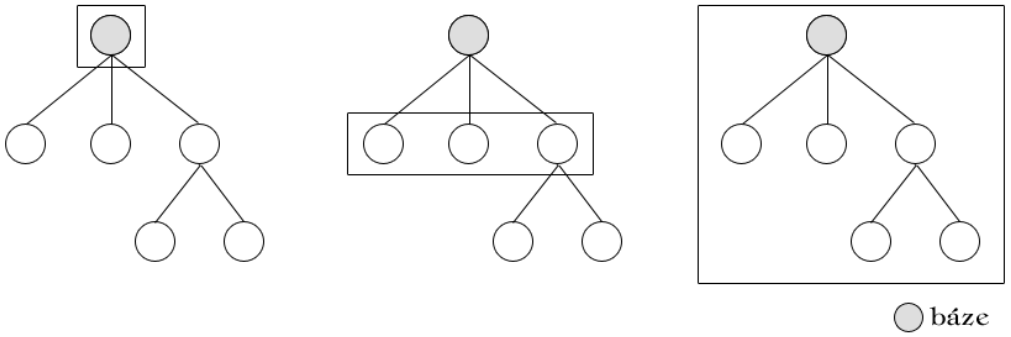
Operace, kterou klient požaduje zrušení předchozí nedokončené operace

Dotazování

search

Nejčastěji využívaná operace sloužící vyhledávání dat v adresáři. Vyhledávání probíhá pomocí následujících parametrů:

- *báze* - rozlišovací jméno objektu, ve kterém ve stromové struktuře začíná prohledávání
- *rozsah* - rozsah prohledávání vzhledem k zadanému bázovému objektu:
  - `BASE` - prohledá se bázový objekt
  - `ONE` - prohledají se objekty pouze o jednu úroveň níž vzhledem k zadanému bázovému objektu (bázový objekt se neprohledává)
  - `SUBTREE` - prohledá se celá větev od bázového objektu, včetně něj



Obrázek 2: Znázornění rozsahů při vyhledávání v LDAP

- *filtr* - definuje kritéria, podle kterých jsou vybírány položky pro vrácení. Filtry jsou definovány pomocí dané syntaxe a sémantiky.

  Základní syntaxe filtru je:

  `<atribut><operátor><hodnota>`

  Pomocí logických operátorů lze filtry libovolně kombinovat. Využívá se prefixové notace.

  (`<operátor>` (`<filtr1 >`)(`<filtr2 >`)`...`(`<filtr`n `>`))

Operátor	Popis
=	Rovná se
~=	Přibližně se rovná
<=	Lexikograficky menší než nebo rovno
>=	Lexikograficky větší než nebo rovno
&	AND
|	OR
!	NOT

Tabulka 3: Seznam operátorů pro definování filtrů

Při vytváření filtrů je využíváno také znaku *, zastupující libovolný znak nebo posloupnost znaků. Vše bude názorně zobrazeno na následujících příkladech.

`cn=vas186`
Vyhledá položky, jejichž hodnota atributu `cn` je rovna hodnotě vas186

`(!(email=*))`
Vyhledá položky, které nemají vyplněny hodnotu emailu.

`(&(objectClass=user)(sn=Vas*)(!(givenName=David))`
Vyhledá uživatele, jejichž příjmení začíná na Vas, ale nemají křestní jméno David

- *požadované atributy* - vrácené výsledky je možno omezit pouze na atributy, které pro nás mají význam
- *limity* - prohledávání lze omezit na maximální počet vrácených položek nebo lze zadat maximální dobu trvání vrácení výsledků. Tyto limity je vhodné použít, nemáme-li představu o možném množství vrácených dat.
  - `sizeLimit` - maximální počet vrácených položek
  - `timeLimit` - maximální doba zpracování výsledků

compare

porovná hodnotu atributu konkrétní položky se zadanou hodnotou. Vrací logickou hodnotu

Aktualizace

add

Přidá nový záznam se zadanými hodnotami do adresáře.

modify

Upraví atributy záznamu. Umožňuje přidávat, upravovat či odebírat atributy existujícího objektu.

modify RDN

Slouží ke změně rozlišovacího jména položky. Tedy v podstatě k přemístění objektu ve stromové struktuře. Dovoluje však změnit pouze relativní rozlišovací jméno v rámci větve, ve které je položka umístěna.

delete

Odebrání záznamu z adresáře. Odebrat lze pouze listy ze stromové struktury. Tedy nelze odebrat objekty obsahující další záznamy.

3.1.4 Bezpečnostní model

Úkolem bezpečnostního modelu je zabránit přístupu neoprávněné osoby k informacím v adresáři.

Autentizace

Úkolem autentizace je ověření identity uživatele, který se snaží o spojení s adresářovým serverem. Uživatel může být autentizován následujícími způsoby:

1. **Anonymní autentizace** – v rámci operace `bind` nejsou serveru předávány žádné informace o uživateli přistupujícímu k adresáři. V případě anonymní autentizace má však uživatel přístup pouze ke čtení veřejných atributů.

2. **Jednoduchá autentizace** – pomocí DN uživatele přistupujícího k adresáři a jemu příslušného hesla v čistém textu.

3. **SASL mechanismy** – Simple Authentication and Security Layer [5] je standardizovaná cesta pro dodatečné zabezpečení spojově orientovaných komunikačních protokolů. Jedná se v podstatě o jasně definované rozhraní, jehož prostřednictvím lze propojovat standardní autentizační mechanismy s komunikačními protokoly. Disponuje několika druhy ověřovacích mechanismů řídících výzvy a odpovědi mezi klientem a serverem:

   - PLAIN – nejjednodušší mechanismus pomocí uživatelského jména a hesla
   - DIGESTMD5 – mechanismus autentizace pomocí zahashovaného hesla
   - GSSAPI – autentizace pomocí mechanismu Kerberos
   - SKEY – mechanismus pro ověřování pomocí jednorázových hesel
   - EXTERNAL, ANONYMOUS, NTLM...

   Základní schéma použití SASL autentizace je následující:

   - klient předá informace pro autentizaci
     - DN přistupujícího uživatele
     - zvolený mechanismus

  - credentials – data prokazující identitu v závislosti na příslušném mechanismu
- přes LDAP API dojde autentizační požadavek LDAP serveru, který pro jeho vyřízení použije příslušný autentizační modul dle zadaného mechanismu
- příslušný autentizační modul na základě předaných dat rozhodne, zda je identita prokázána či nikoli a uživateli buď povolí nebo zakáže přístup

V případě jednoduché autentizace na adresářovém serveru proběhne úspěšně tehdy, když dané uživatelské jméno bude připadat existujícímu záznamu v adresáři, ve kterém se rovněž shoduje hodnota zadaného hesla s hodnotou uloženou v atributu `userPassword`.

Podobným způsobem probíhá i autentizace správce celého adresáře, jehož význačné jméno definujeme v konfiguračním souboru implementace adresářového serveru. Jen s tím rozdílem, že danému uživateli přísluší heslo, které není uchováno v adresáři, ale ve zmíněném konfiguračním souboru.

To sebou přináší i nutná bezpečnostní opatření. Jelikož hesla jsou velmi citlivá a nepřejeme si jejich zveřejnění, můžeme v adresářovém serveru využívat možností jejich ukládání v šifrované podobě. Implementace adresářových serverů nejednou obsahují programy pro šifrování hesel pomocí známých algoritmů. Veškerá hesla je tedy doporučeno ukládat v šifrované podobě. Tedy jak atributy userPassword, tak i zmiňované heslo správce adresáře v konfiguračním souboru.

Autorizace

Autorizace úzce souvisí s mechanizmem autentizace a nastupuje po jejím úspěšném ukončení. Jejím úkolem je zajistit, aby měl autentizovaný uživatel přístup pouze k datům a operacím, ke kterým je oprávněn. Tato problematika tedy úzce souvisí s nastavením přístupových práv k záznamům a atributům.

Přístupová práva můžou být nastaveny až na úroveň jednotlivých atributů. Specifikace práv jsou uvedeny v tzv. Access Control Listech (ACL) v konfiguračním souboru daného adresáře a definují, který uživatel, skupina uživatelů či nějaký jiný objekt má jaká práva k různým atributům a operacím. Určují například, do kterých atributů je možno zapisovat, které je možno pouze číst apod.

Zabezpečení

Zabezpečení autentizace uživatele i veškerou ostatní komunikaci mezi uživatelem a adresářem lze zabezpečit pomocí protokolu SSL (Secure Socket Layer) nebo jeho následníka TLS (Transport Layer Security), které slouží primárně pro zabezpečení TCP/IP komunikace.

3.1.5 LDAP Data Interchange Format (LDIF)

LDIF [6] je standardizovaný textový formát pro výměnu adresářových dat. Jedná se o jednoduchou textovou reprezentaci záznamů v adresáři. Každý záznam je v něm definován svým rozlišovacím jménem DN a výčtem atributů s odpovídajícími hodnotami. U každého záznamu nesmíme zapomenout uvést, jaké objektové třídy, popřípadě tříd, je záznam instancí. Pomocí souborů LDIF můžeme jednoduše importovat do adresáře množství nových záznamů nebo naopak můžeme celý obsah adresáře do souboru tohoto formátu vyexportovat. Níže je uveden příklad výpisu jedné položky z adresáře. Konkrétně se jedná o záznam s jedinečným jménem `cn=vas186,o=6`. Můžeme vidět, ze kterých tříd je záznam instanciován a hodnoty veřejných atributů z těchto tříd.

```
dn: cn=vas186,o=6
loginShell: /bin/ldap-sh
homeDirectory: /home/fei/vas186
gidNumber: 10000
uidNumber: 41616
mail: david.vastik.st@vsb.cz
uid: vas186
givenName: David
fullname: David Vastik
messageServer: cn=HOMEN,ou=HOME,o=CVT
sn: Vastik
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: Person
objectClass: Top
objectClass: ndsLoginProperties
objectClass: posixAccount
objectClass: TUOChipCardClass
ndsHomeDirectory: cn=HOMEN_FEI,ou=HOME,o=CVT#4#HOME\vas186
groupMembership: cn=UZIVATELE,o=GROUPS
groupMembership: cn=STU_FEI,ou=FEI,o=GROUPS
groupMembership: cn=STU_VSB,o=GROUPS
cn: vas186
```

3.1.6 Directory Service Markup Language (DSML)

Ve snaze o využití XML v adresářových službách vznikl nástroj pro reprezentaci dat z adresářového serveru nazvaný Directory Service Markup Language [7], který by mohl formát LDIF v budoucnu nahradit. Vznikl za účelem lepšího přenosu adresářových dat pomocí internetových protokolů. To je výhodné například pro bezpečný přenos dat nad transparentními protokoly. Další výhodou je to, že formát XML je pro webové aplikace lépe zpracovatelný než textový formát. Pro ukázku je uveden příklad s několika atributy z ukázky ve formátu LDIF.

```
<dsml:entry dn="cn=vas186,o=6">
  <dsml:objectclass>
    <dsml:oc-value>top</dsml:oc-value>
    <dsml:oc-value>person</dsml:oc-value>
    <dsml:oc-value>organizationalPerson</dsml:oc-value>
    <dsml:oc-value>inetOrgPerson</dsml:oc-value>
  </dsml:objectclass>
  <dsml:attr name="sn">
   <dsml:value>Vastik</dsml:value>
  </dsml:attr>
  <dsml:attr name="uid">
   <dsml:value>vas186</dsml:value>
  </dsml:attr>
  <dsml:attr name="mail">
   <dsml:value>david.vastik.st@vsb.cz</dsml:value>
  </dsml:attr>
  <dsml:attr name="givenName">
  <dsml:value>David</dsml:value>
  </dsml:attr>
  <dsml:attr name="cn">
   <dsml:value>vas186</dsml:value>
  </dsml:attr>
</dsml:entry>
```

V současné verzi 2.0 je možno pomocí tohoto nástroje také provádět příkazy na adresářovém serveru. Například odebrání záznamu:

```
<delRequest dn="cn=vas186,o=6" />
```

Po provedení takovéhoto příkazu je samozřejmé i návratová hodnota vrácena ve formátu XML. Při pokusu o odebrání neexistujícího záznamu by mohla vypadat takto:

```
<delResponse matchedDN="cn=vas186,o=6">
  <resultCode code="32" descr="noSuchObject">
  <errorMessage>DSDEL::230234<errorMessage>
</delResponse>
```

3.1.7 Implementace LDAP serveru

Novell eDirectory

Adresářová služba eDirectory [8] přišla na scénu se systémem Novell NetWare 5.1 v roce 2000 jako alternativa k jejímu předchůdci NDS (Novell Directory Services). V následujících systémech NetWare 6 a NOES (Novell Open Enterprise Server) již představuje jejich standardní systémový adresář. Jeho kvalitu dokazuje i to, že splňuje veškeré podmínky pro adresáře LDAPv3.

Zřejmě nejvýznamnějším rysem eDirectory je její multiplatformnost. Je možné ji implementovat a používat jak v původním prostředí NetWare, tak i na jiných operačních platformách. V současnosti se jedná konkrétně o SUSE Linux, Red Hat Linux, Solaris, AIX, HP-UX a Windows NT/2000/2003 Server. eDirectory se tak dostala se do role ojedinělého prostředku, jehož prostřednictvím lze realizovat jednotnou centrální správu heterogenních sítí. Konkrétním příkladem této skutečnosti je heterogenní síťová platforma NOES, jejíž základnu tvoří systémy Novell NetWare 6.5 a SUSE LINUX Enterprise Server 9.

Novell poskytuje webovou aplikaci, která umožňuje snadno a rychle vyhledávat informace z eDirectory zvanou eGuide. Ta je velmi efektivním nástrojem pro vyhledávání adres, telefonních čísel nebo jiných informací o uživatelích uložených v adresáři. Uživatelé s odpovídajícími právy mají možnost uložené informace také upravovat. To vše bez nutnosti instalace jakéhokoliv klientského softwaru.

Sun Directory Server [9]

Jedno z nejúspěšnějších vysoce dostupných adresářových řešení, založených na LDAP. Jedná se o otevřené řešení podporující DSML v2.0. a velké množství platforem podobně jako eDirectory. Také splňuje podmínky LDAPv3 a implementuje mnoho opatření k dosažení nejvyšší bezpečnosti.

OpenLDAP

Projekt OpenLDAP [10] vznikl týmem lidí, jejichž záměrem bylo vyvinout robustní, komerčních produktů dosahující kvalit, ale hlavně volně šiřitelnou implementaci LDAP serveru. Prvotní jádro tvořili pouzí tři členové, postupem času, se však na projektu začala podílet spousta uživatelů.

4 Správa identit

Zatímco ještě před pár lety bylo naprosto běžné, že každá aplikace a každý podnikový systém řešil správu uživatelů a zabezpečení jejich přístupu samostatně, v poslední době se situace radikálně mění. Do podnikových sítí je nasazována tzv. správa identit (identity management, IDM) zajišťující efektivní centrální správu veškerých účtů v informačních systémech. Ta se stává nezbytným předpokladem moderního způsobu komunikace a přístupu k informacím.

Hlavními přínosy identity managementu je zvýšení bezpečnosti podnikové sítě a uživatelů do ní přistupujících, synchronizace uživatelských účtů mezi všemi aplikacemi a systémy či výrazné ulehčení práce administrátorů počítačové sítě. Vše je dáno automatizovanými procesy identity managementu například při správě uživatelských účtů a s nimi spojené řízení přidělování přístupů do systémů. Třeba při přijmutí nového zaměstnance dojde po prvotním vstupu informací o jeho osobě do personálního informačního systému k vygenerování veškerých náležitostí, uživatelských jmen, hesel, práv a rolí uživatele, dle předem nastavených a určených pravidel. Systém navíc umožňuje mít v každém okamžiku přesný snímek o uživatelských přístupech, rolích a právech jednotlivců.

4.1 Novell Identity Manager

Systémy pro správu identit poskytuje většina světových firem zabývající se vývojem softwaru, jako IBM, Sun, Oracle apod. Jelikož VŠB využívá řešení od firmy Novell, budu se konkrétně zabývat hlavně tímto produktem nazvaným Novell Identity Manager [11]. Ten lze, stejně jako i ostatní řešení nasadit na adresářové služby, portály, databáze a řadu různých aplikací. Systém je nezávislý na platformě a může tedy přistupovat k různým operačním systémům. Podporuje spoustu známých protokolů, pomocí kterých může být na tyto různé platformy či aplikace různých výrobců nasazen a nabízí následující funkce:

- automatizovaný proces obsluhy
- správa hesel
- samoobslužnost uživatelů
- automatizované toky úkolů
- správa založená na rolích
- delegovaná správa
- sledování událostí v celém systému a tvorba reportů
- seznam uživatelů v organizaci a organizační schémata
- grafické konfigurační nástroje

Okamžitý přístup ke zdrojům

Při použití Novell Identity Manageru mohou být noví uživatelé okamžitě produktivní díky tomu, že obdrží přístup k potřebným zdrojům hned první den jejich pracovního poměru v organizaci. Při vytvoření nového uživatele v databázi lidských zdrojů jsou všechny jeho účty generovány automaticky podle pravidel organizace a všechny nezbytné schvalovací procesy jsou automatizovány, tudíž někdy několikadenní čekání může být zkráceno do několika minut.

Správa hesel

Novell Identity Manager umožňuje synchronizovat hesla uživatele a poskytnout mu tak jediné heslo pro přístup ke všem systémům. Uživatel si tak nemusí pamatovat různá hesla ke všem systémům, do kterých přistupuje, ale pouze ono jediné. V Identity Manageru lze zajistit, aby hesla, která si uživatelé vytvářejí, byla bezpečná. Pro hesla lze totiž vytvářet a uplatňovat silné politiky platné v celém systému. To vše pomůže zvýšit ochranu celé společnosti před útoky zaměřenými na hesla.

Samoobslužnost uživatelů

Jednoduché operace při automatizované správě zvládnou sami uživatelé. Bylo zjištěno, že v některých organizacích se 30% požadavků na helpdesk týkalo pouze zapomenutého hesla do různých aplikací. To i spousta dalších manuálních operací, kterými se IT odborníci museli zabývat samozřejmě s nasazením identity managementu odpadá. Identity Manager tímto umožňuje organizacím zvýšit efektivitu zároveň se snížením nákladů.

Správa založená na rolích

Uživatelům jsou přidělovány zdroje na základě rolí a definovaných politik.

iManager

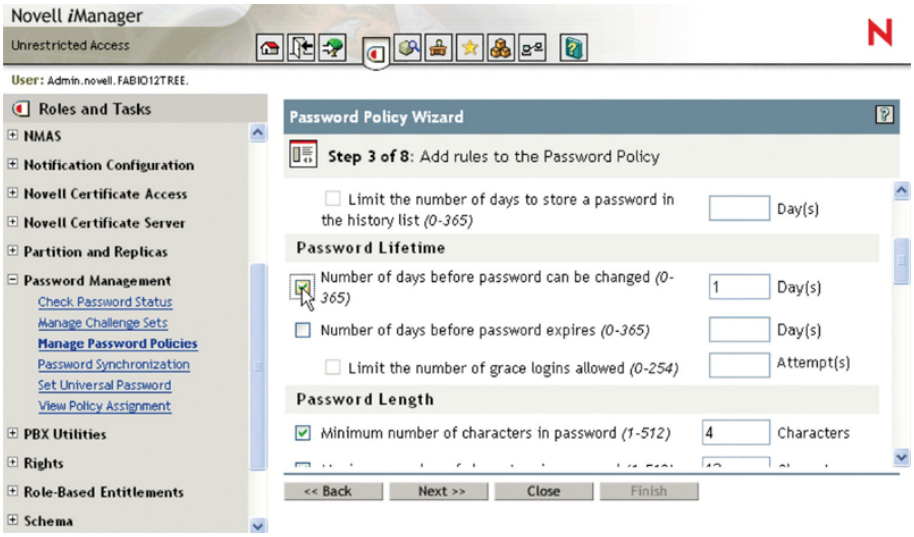
iManager poskytuje rozhraní pro přístup k několika produktům, mezi něž patří i Identity Manager. Umožňuje tedy přes webový prohlížeč z libovolného místa centrálně spravovat identity, uživatelské role, přístupová práva nebo například politiku hesel. V ukázce na obrázku 3 [11] je vidět právě nastavování pravidel pro vytváření uživatelských hesel.

Designer for Identity Manager

Mocným nástrojem pro konfiguraci Identity Manageru je Designer, grafický nástroj určený například pro definování procesů při správě identit. Umožňuje také třeba definování schvalovacích postupů či vytváření a testování scénářů při definování politik ještě před uvedením do provozu. Můžeme tak pomocí něj otestovat chování systému v různých situacích. Vše bez nutnosti kódování či skriptování. Významnou měrou tedy usnadňuje implementaci a konfiguraci Identity Manageru. Dokáže také generovat dokumentaci k celému projektu.

Centrální úložiště identit

Novell Identity Manager je provázaný s eDirectory, adresářovým serverem Novellu, který slouží jako spolehlivé úložiště veškerých identit. Správce identit od Novellu obsahuje



Obrázek 3: Novell iManager

všechny důležité funkce a nástroje v nejlépe zpracovaném uživatelském rozhraní ze všech systémů. Správa identit probíhá na základě předem stanovených pravidel, která umožňují administrátorům nastavit vztahy mezi aplikacemi a řídit tok dat. Komunikace mezi úložištěm identit a aplikacemi je založena na XML.

4.2 Sun Identity Manager

Srovnatelným řešením pro správu identity je produkt firmy Sun Microsystems [12]. Produkt má propracovanou správu uživatelských rolí. Ty jsou rozděleny do několika kategorií jako „Business Role“ nebo „IT Role“. Veškeré role lze různě aktivovat či deaktivovat, dokonce třeba pouze na omezenou dobu. Pro jejich správu Sun disponuje výborným nástrojem s uživatelským rozhraním zvaným Role Manager. Ten umožňuje importovat předem nadefinované obchodní role i automatizovat přidělování a audit účtů na úrovni obchodních rolí. Další výhodou tohoto řešení je propracovaná podpora exportu dat týkajících se identit do externích datových skladů pro další zpracování.

Pro přístup k IDM je možno využít webové aplikace, která administrátorům umožňuje provádět operace týkající se správy uživatelských účtů, jako je jejich vytváření nebo modifikace. Samotným uživatelům poskytuje prostředí pro změnu hesla a podobně. Třeba i pro změnu osobních informací, pokud k tomu mají oprávnění.

Business Process Editor

Nedílnou součástí produktu je také Business Process Editor, vývojářský nástroj pro konfiguraci správy identit, definování automatizovaných procesů či politik. Také umožňuje navrhovat veškeré webové formuláře. Editor lze spustit i ze vzdálené stanice.

4.3 Nasazení IDM na VŠB-TUO

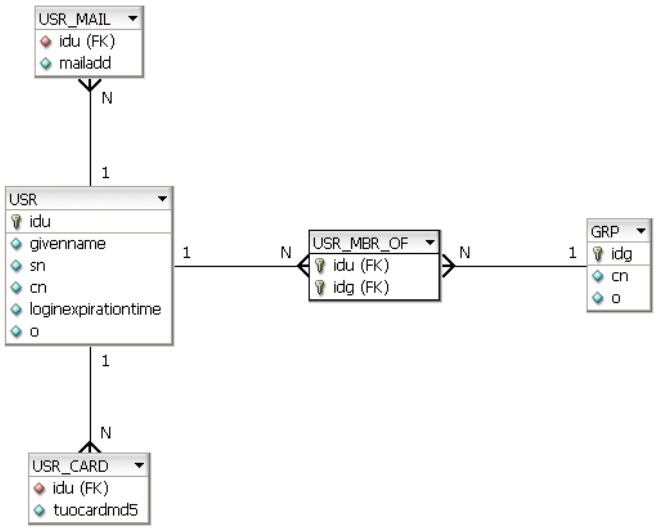
Před nasazením identity managementu bylo nutno provést důkladnou analýzu aktuálního stavu různých programových prostředků, do kterých IDM bude či nebude integrován. Bylo třeba definovat, jaké údaje budou nezbytné pro popisování identit, v jakých rolích mohou v systému vystupovat a podobně.

Na VŠB – TU Ostrava proběhlo nasazení identity managementu v heterogenním počítačovém prostředí školy zejména pro účely automatizace správy uživatelských identit a zajištění aktuálnosti dat mezi jednotlivými systémy [13].

K popsání identity uživatele v databázi slouží následující tabulky:

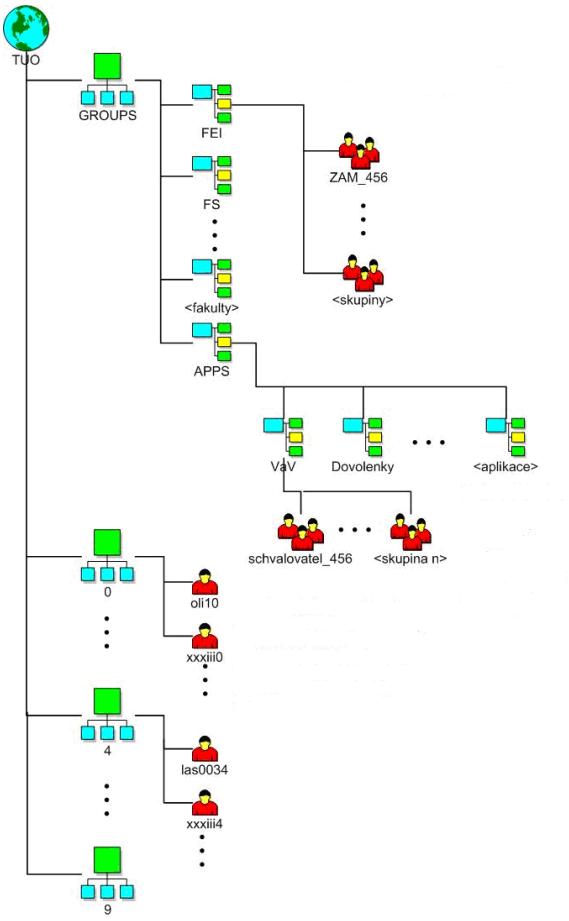
- `USR` – tabulka se základními atributy o uživateli
- `USR_MAIL` – tabulka obsahující uživatelské e-maily
- `USR_CARD` – tabulka obsahující uživatelské e-maily
- `GRP` – tabulka skupin
- `USR_MBR_OF` – spojovací tabulka definující příslušnosti uživatelů ke skupinám

ER diagram znázorňující vazby mezi popsanými tabulkami včetně nejdůležitějších atributů vzhledem k porovnávání je na obrázku 4.



Obrázek 4: Struktura databáze

Struktura adresáře je znázorněna na obrázku 5. Můžeme vidět, že uživatelé nejsou rozděleni do skupin podle fakult a útvarů. To z důvodu těch, že studenti mohou studovat na více fakultách zároveň nebo zaměstnanci mohou pracovat na více útvarech. Veškeří uživatelé tak jsou rozděleni do deseti kontejnerů pojmenovaných číslicemi od 0 do 9. Podle poslední číslice uživatelského jména jsou poté záznamy ukládány do těchto kontejnerů. Skupiny jsou poté uspořádány podle fakult a příslušnost uživatelů k těmto skupinám je rozeznávána podle atributů členství ve skupině.



Obrázek 5: Struktura adresáře

Přehled mapování

V následujících tabulkách jsou uvedeny mapování atributů mezi relační databází a adresářovou službou.

DB2 (indirect.usr)	LDAP (class user)
idu	uidNumber
givenName	givenName
sn	sn
	fullname
	groupMembership
	securityEquals
	loginDisabled
cn	cn
mail	mail
TUOCardMD5	TUOCardMD5
loginExpirationTime	loginExpirationTime
o	
ou	
cn	uid (uniqueID)
	gidNumber
	homeDirectory
	loginShell

Tabulka 4: Mapování atributů pro uživatelské účty

DB2 (indirect.grp)	LDAP (class group)
idg	gidNumber
cn	cn
o	
ou	
	groupMembership

Tabulka 5: Mapování atributů pro skupiny

DB2 (indirect.usr_mbr_of)	LDAP (class group)
idu	member
idg	groupMembership

Tabulka 6: Mapování atributů pro členství ve skupině

5 Aplikace porovnávání dat

V následující části bude popsán návrh aplikace pro porovnávání dat s následným popisem vlastní implementace o technologiích při implementaci využívaných.

5.1 Zadání, specifikace

Zadání projektu bylo v původní fázi velmi obecné. Byl zadán pouze hlavní cíl, tedy porovnání dat mezi relační databází a adresářovou službou a následné vyobrazení nekorespondujících dat. Detailnější specifikace aplikace tak vznikala během konzultací se zadavatelem. Aplikace však měla být navržena pro obecné porovnávání. Tedy aby si uživatel mohl sám dynamicky nastavovat atributy, které se budou porovnávat, tabulky, ve kterých se atributy vyskytují, vytvářet mapování atributů a definovat další nastavení spojené se samotným porovnáváním. Program tedy měl kromě přehledu výsledků poskytnout také prostředí pro konfiguraci těchto parametrů. Průběh nejdůležitější funkce programu, tedy samotné porovnávání je zachyceno pomocí sekvenčního diagramu na obrázku 6 na straně 30.

5.2 Funkční analýza

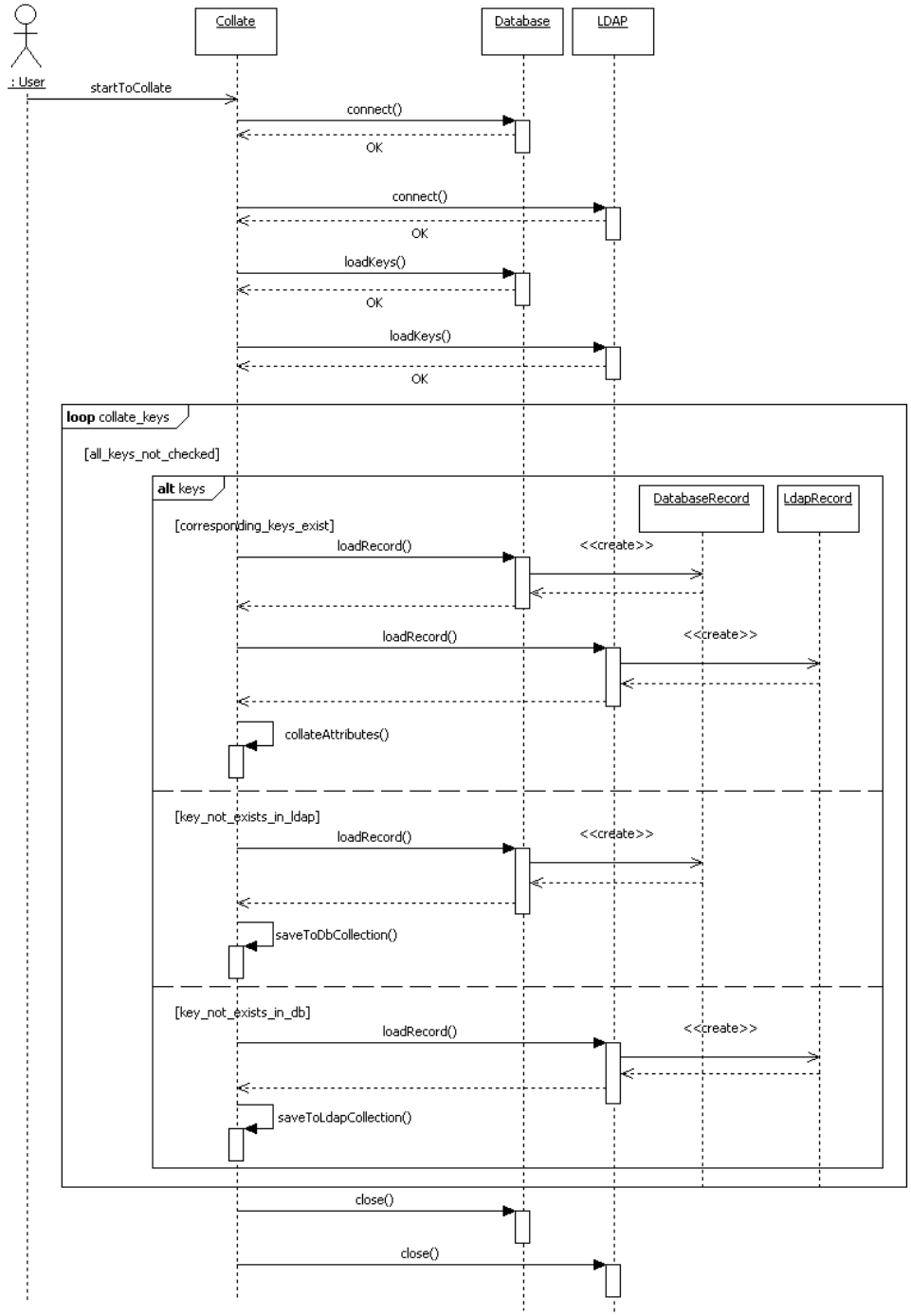
Funkční požadavky

Seznam funkčních požadavků jsem tedy podle specifikace shrnul do pěti obecných bodů

- nastavení porovnávání
  - správa mapování
  - správa databázových a LDAP atributů
  - správa databázových tabulek
  - další nastavení spojené s porovnáváním
- porovnávání dat (jednotné pro porovnávání uživatelů i skupin)
- vyobrazení výsledků porovnávání
- filtrování výsledků
  - filtrování záznamů nalezených pouze v databázi nebo v adresáři
  - filtrování záznamů s nekorespondujícími hodnotami mapovaných atributů
- vyhledávání ve výsledcích

Nefunkční požadavky

Jedním z nefunkčních požadavků bylo to, aby systém běžel jednotně jak pod operačním systémem Microsoft Windows, tak pod operačním systémem Linux. Zadání nekladlo

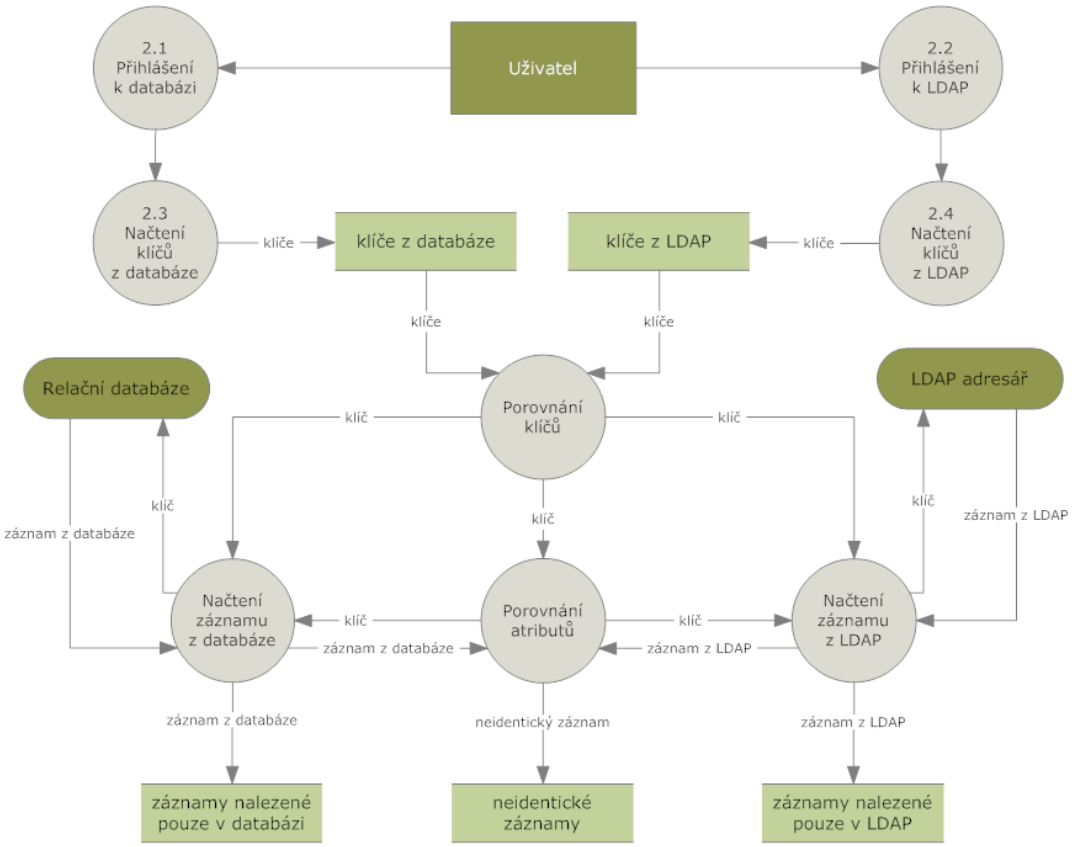


Obrázek 6: Sekvenční diagram pro průběh porovnávání

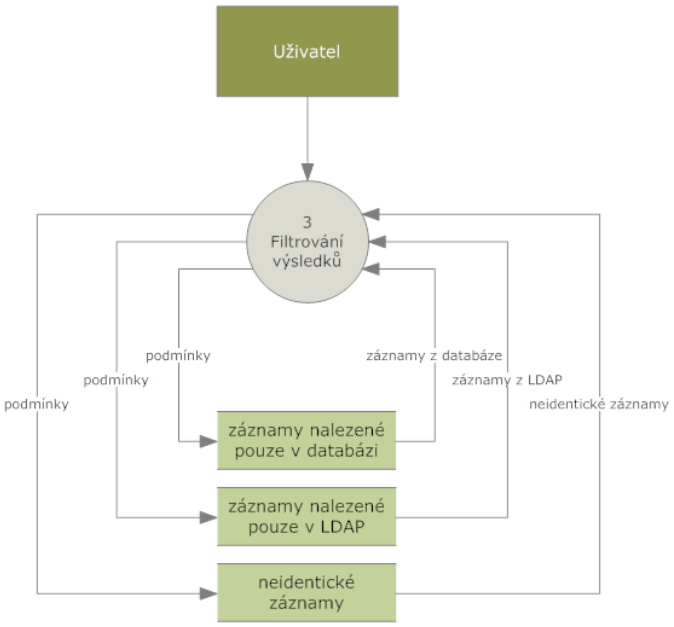
omezení na programovací jazyk, zvolil jsem tedy jazyk Java, z důvodu jeho dobré podpory práce s databázemi a vzdálenými objekty obecně. Hlavním kritériem pro výběr Javy však byla její přenositelnost. Dalším nefunkčním požadavkem bylo, aby aplikace dokázala porovnávat data uložená v databázích DB2 a MySQL.

Diagramy datových toků

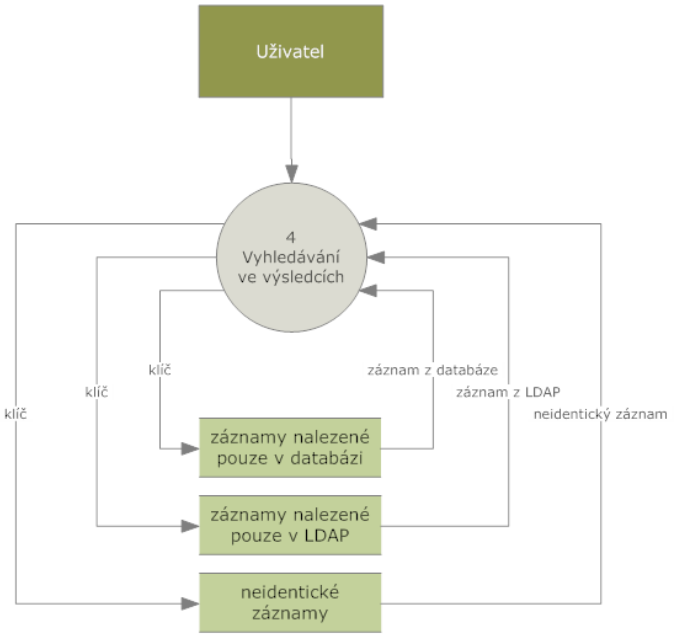
Níže jsou uvedeny diagramy pro funkce systému, které mají význam ve smyslu znázornění datových toků a úložišť.



Obrázek 7: DFD – Porovnávání



Obrázek 8: DFD – Filtrování výsledků



Obrázek 9: DFD – Vyhledávání ve výsledcích

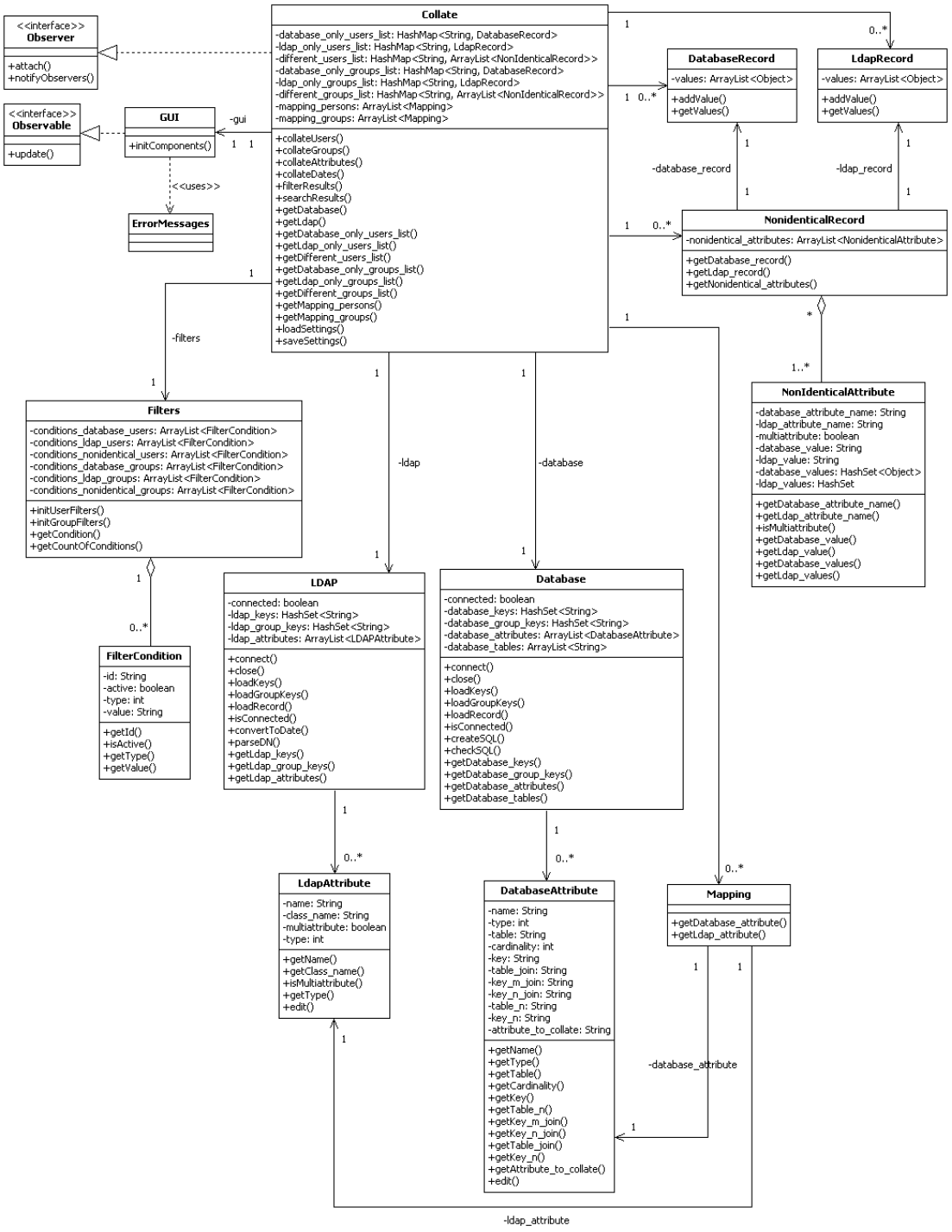
5.3 Datová analýza

V této části jsou stručně slovně popsány všechny navržené třídy aplikace. Na obrázku 10 je zobrazen třídní diagram zachycující vztahy mezi těmito třídami.

- `Collate` je hlavní třída aplikace, ve které je implementováno samotné porovnávání záznamů. Uchovává nalezené nekorespondující záznamy, obsahuje tedy i metody spojené s filtrováním a vyhledáváním výsledků. Ve třídě je také implementována práce s veškerým nastavením pro porovnávání. Tedy načítání a ukládání nastavení z nebo do souboru či metody pro kontrolu správnosti vytvářených mapování, atributů a podobně.
- `Database` je třída využívaná pro práci s relační databází. Pomocí ní se můžeme připojit k databázi a načíst si z ní záznamy pro porovnávání uživatelů nebo skupin. Obsahuje také metody pro vytvoření SQL příkazu pro načítání mapovaných atributů a kontrolu správnosti tohoto příkazu.
- `Ldap` je třída využívaná pro práci s adresářovou službou. Obdobně jako u třídy Database se pomocí ní připojujeme k adresářové službě a načítáme z ní potřebné záznamy.
- `DatabaseAttribute` uchovává informace o vytvořeném databázovém atributu. Jedná se o informace o názvu a SQL typu atributu, kardinalitě a případných tabulkách, se kterými je atribut provázán či klíčích, přes které je s těmito tabulkami propojován. Atributy mohou být následujících typů:
  - numerické typy – SHORT, INT, BIGINT, FLOAT, DOUBLE
  - textové typy – CHAR, VARCHAR, TEXT
  - datumové typy – DATE, TIMESTAMP
- `LdapAttribute` uchovává informace o vytvořeném LDAP atributu. Opět se jedná o název, datový typ, zda atribut vrací pouze jednu či více hodnot, tedy jedná-li se o multiatribut. LDAP primárně vrací hodnoty v textové podobě. Aplikace se je po načtení snaží na zvolený datový typ převést.
  - Integer – převod na datový typ Integer
  - String – tyto hodnoty samozřejmě nepřevádí, pouze si pamatuje, zda má při porovnávání brát ohled na malá a velká písmena či nikoli
  - Date – načtený řetězec se snaží převést na datový typ Date podle definované struktury
  - DN – speciální případ, kdy vrácená hodnota je distinguished name. Taková hodnota je automaticky „ořezána“ pouze na hodnotu common name
- `Mapping` reprezentuje vytvořená mapování. Obsahuje instance tříd DatabaseAttribute a LdapAttribute. Tedy uchovává informace o namapovaných atributech.

- `DatabaseRecord` uchovává hodnoty jednoho záznamu načteného z databáze.
- `LdapRecord` uchovává hodnoty jednoho záznamu načteného z LDAP adresáře.
- `NonidenticalRecord` uchovává informace o záznamu, u kterého byly během porovnávání atributů zjištěny rozdílné hodnoty. Obsahuje instance tříd DatabaseRecord a LdapRecord plus množinu tříd NonidenticalAttribute uchovávající informace o tom, které atributy se v záznamech liší a hodnoty těchto atributů.
- `NonidenticalAttribute` jak již bylo zmiňováno, obsahuje informace o atributech, ve kterých byly zjištěny rozdílné hodnoty.
- `Filters` uchovává seznamy podmínek, pomocí kterých jsou filtrovány výsledky jednotlivých kategorií.
- `FilterCondition` uchovává informace o podmínce využívané při filtrování výsledků. Obsahuje například informace o tom, o jaký typ podmínky se jedná či hodnotu, se kterou jsou při filtrování veškeré záznamy porovnávány.
- `GUI` reprezentuje grafické rozhraní celé aplikace. Poskytuje prostředí pro konfiguraci a přehledně zobrazuje průběh porovnávání či přehled výsledků s možnostmi filtrování a vyhledávání. Samozřejmě umožňuje zobrazit detailní popis nalezených nekorespondujících záznamů.
- `ErrorMessages` uchovává textový popis chyb, které mohou nastat při konfiguraci aplikace nebo při samotném porovnávání dat či následné práci s výsledky. Tuto třídu využívá grafické rozhraní pro výpis při nastalých chybách.

Při implementaci je pro realizaci správného zobrazení průběhu porovnávání v grafickém prostředí využíváno známého, hojně využívaného návrhového vzoru Observer-Observable.

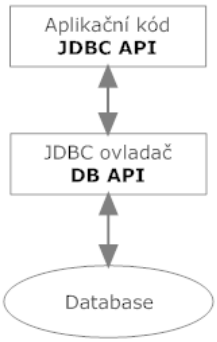


Obrázek 10: Třídní diagram

5.4 Implementace

5.4.1 Práce s databází v Javě

Při práci s databází jsem využíval balíku tříd JDBC API [14]. Tedy rozhraní pro přístup k relačním datovým zdrojům, vyvinuté společností SUN Microsystems. Aplikace měla být optimalizována pro databáze DB2 a MySQL. S rozhraním JDBC to však nebyl příliš velký problém, protože to umožňuje univerzální přístup k široké škále relačních databází. JDBC totiž obsahuje Interface, který musí všichni dodavatelé databází a k nim odpovídajících ovladačů JDBC implementovat. Mohl jsem se tím pádem soustředit zejména na logiku aplikace a nemusel řešit mechanismy připojení k databázi, různé druhy zpracování výsledků a podobně. Tyto věci řeší ovladače pro DB2 a MySQL. Základní princip komunikace mezi aplikací a databází je znázorněn na obrázku 11. Samotná práce s databází probíhá pomocí základních tříd a rozhraní z balíků: `java.sql` a `javax.sql`. V práci jsem si však vystačil s první zmiňovanou. Budou-li v následující části textu uváděny názvy tříd, budou automaticky chápány jako třídy z knihovny `java.sql`.



Obrázek 11: Zjednodušené schéma komunikace mezi aplikací a databází pomocí JDBC ovladače

Pro získání dat z databáze je nutno provést pět následujících kroků:

1. **Připojení k databázi**

   V této části je potřeba v první řadě vybrat ovladač, pomocí kterého budeme pracovat s databází. Pro účely mé práce jsem tedy potřeboval jak ovladač pro DB2, tak ovladač pro MySQL. Pomocí těchto ovladačů nebo přesněji pomocí správce ovladačů reprezentovaného třídou `DriverManager` poté získáme spojení s databází ze zadané URI. Získané spojení bude uchováno v instanci třídy `Connection`, kterou budeme v další fázi využívat. Formát URI je následující:

   ```
   jdbc:<typ ovladače>://<host>:<port>/<název databáze>
   ```

   U MySQL může za názvem databáze následovat ještě seznam parametrů uveden za znakem otazníku. Parametry jsou poté oddělovány znakem `&`. Např.:

   ```
   ?characterEncoding=utf8&zeroDateTimeBehavior=convertToNull
   ```

Součástí připojení k databázi je samozřejmě také autentizace uživatele.

2. **Příprava SQL příkazu**

   Po úspěšné inicializaci ovladače a navázání spojení je potřeba připravit příkaz, který bude využit pro získání dat z databáze. Pro reprezentaci takovýchto příkazů existují následující třídy:

   - `Statement` - pro volání statických příkazů
   - `PreparedStatement` - pro volání připravených příkazů s parametry
   - `CollableStatement` - pro volání uložených procedur

   V práci jsem využíval první zmiňovanou pro zjišťování počtu záznamů a načítání klíčů z databáze. Druhou poté pro načítání jednotlivých záznamů, kde byly jako parametry postupně předávány právě načtené klíče. Instance těchto tříd získáme voláním metod `createStatement()` a `prepareStatement()` z objektu získaného spojení.

3. **Volání SQL příkazu**

   Vytvořený SQL příkaz nyní pošleme ke zpracování do databáze. K tomu nám slouží následující metody:

   - `execute()` - obecný příkaz SQL
   - `executeQuery()` - pro SQL příkazy typu SELECT, vracející z databáze sadu vyhovujících záznamů
   - `executeUpdate()` - pro SQL příkazy typu INSERT, DELETE, UPDATE a další příkazy pro úpravy na databázi
   - `executeBatch()` - pro spuštění dávky SQL příkazů

   Jelikož jsem pro potřeby porovnávání dat potřeboval pouze číst z databáze, vystačil jsem si s metodou `executeQuery()`.

4. **Zpracování vrácených výsledků**

   Po volání vytvořeného dotazu nám databáze vrátí sadu výsledků reprezentovanou třídou `ResultSet`, ze které už snadno pomocí jednoduchých metod získáme potřebná data.

   Při čtení dat pro potřeby porovnávání jsem se rozhodl jít cestou menšího počtu přístupů do databáze na úkor složitějšího zpracování vrácených výsledků. Tedy namísto vytvoření například čtyř SQL dotazů a jejich následné volání a jednodušší zpracování výsledků jsem volil jeden složený, který však z důvodu několika vrácených atributů s možností více hodnot vrátí kartézský součin, který je nutný zpracovat tak, abychom získali pouze potřebné údaje.

```
SELECT indirect.usr.idu, indirect.usr.givenname, indirect.usr.sn, indirect.usr.cn,
    indirect.usr.loginexpirationtime FROM indirect.usr WHERE indirect.usr.cn = '
    vas186';
SELECT indirect.usr_mail.mailadd FROM indirect.usr_mail WHERE indirect.usr_mail.idu =
    41616;
SELECT indirect.usr_card.tuocardmd5 FROM indirect.usr_card WHERE indirect.usr_card.
    idu = 41616;
SELECT indirect.grp.cn FROM indirect.grp INNER JOIN indirect.usr_mbr_of ON indirect.
    grp.idg = indirect.usr_mbr_of.idg WHERE indirect.usr_mbr_of.idu = 41616;
```

Výpis 5: Ukázka více SQL dotazů

```
SELECT indirect.usr.idu, indirect.usr.givenname, indirect.usr.sn, indirect.usr.cn,
    indirect.usr.loginexpirationtime, indirect.usr_mail.mailadd, indirect.usr_card.
    tuocardmd5, indirect.grp.cn
FROM indirect.usr
  INNER JOIN indirect.usr_mail ON indirect.usr.idu = indirect.usr_mail.idu
  INNER JOIN indirect.usr_card ON indirect.usr.idu = indirect.usr_card.idu
  INNER JOIN indirect.usr_mbr_of ON indirect.usr.idu = indirect.usr_mbr_of.idu
  INNER JOIN indirect.grp ON indirect.grp.idg = indirect.usr_mbr_of.idg
WHERE indirect.usr.cn = 'vas186';
```

Výpis 6: Ukázka složeného SQL dotazu

5. **Ukončení připojení k databázi**

   Posledním krokem je ukončení připojení k databázi. A to metodou `close()` na objektu reprezentujícím připojení.

Volání zmiňovaných metod je nutno ošetřit zachytáváním výjimky `SQLException` vyvolané při chybách práce s databází a `ClassNotFoundException` vyvolané při nenalezení třídy ovladače.

Obrázek 12: Průběh práce s databází

V následujícím ukázkovém příkladu je postupně použito všech těchto pět kroků. Jedná se o příklad, který z DB2 databáze ze zadaného zdroje, po úspěšném připojení načte a postupně vypíše seznam všech uživatelských jmen (seznam hodnot uložených v atributu `cn` z tabulky `usr` nacházející se ve schématu `indirect`). Jsou také ošetřeny zmiňované výjimky, které mohou nastat.

```
try {
    String url = "jdbc:db2://db-pretest64.vsb.cz:50001/POKUSNA";
    Class.forName("com.ibm.db2.jcc.DB2Driver");
    Connection con = DriverManager.getConnection(url, "vas186", "password");
    Statement stat = con.createStatement();
```

```
    rs = stat.executeQuery("SELECT indirect.usr.cn FROM indirect.usr;");
    while (rs.next()) {
        System.out.println("cn: " + rs.getString("cn"));
    }
    con.close();
} catch(SQLException e) {
    System.out.println("Chyba při práci s databází");
} catch (ClassNotFoundException e) {
    System.out.println("Ovladač nebyl nalezen");
}
```

Výpis 7: Ukázka práce s databází

5.4.2 Práce s adresářovými službami v Javě

Obdobně jako pro práci s databázemi Java poskytuje rozhraní JNDI [15] pro přístup k adresářovým a jmenným službám. Ani zde nahraje roli typ adresářové služby. Jednotně tedy můžeme pomocí tohoto rozhraní přistupovat ke službám CORBA, RMI, NDS či právě LDAP. Třídy a rozhraní pro práci se jmennými službami najdeme v balíku `javax.naming`. V aplikaci budeme také potřebovat rozšiřující balík `javax.naming.directory` pro přístup k adresářovým službám. Jelikož tyto balíky obsahují poměrně velké množství tříd a rozhraní, popíšu pouze ty, které jsem využíval v aplikaci.

javax.naming

`Context` - Základní rozhraní pro reprezentaci kontextů jmenných služeb. Definuje operace pro vytvoření vazeb jméno-objekt.

`NamingEnumeration` - Rozhraní reprezentující seznam výsledků po vyhledáván.

`NamingException` - Supertřída pro všechny výjimky vyvolané operacemi pro práci se jmennými a adresářovými službami.

`NameNotFoundException` - Výjimka, která je vyvolána, pokud jméno nemůže být rozlišeno, protože není navázáno. Například při pokusu o čtení dat z neexistujícího kontextu.

javax.naming.directory

`DirContext` - Rozhraní reprezentující kontext v adresáři. Definuje metody pro práci s atributy adresářových záznamů. Jedná se zejména o metody:

- `getAttributes()` - pro čtení atributů
- `modifyAttributes()` - pro vytváření, odebírání a úpravy atributů a jejich hodnot

Toto rozhraní také obsahuje metody pro vyhledávání v adresáři. Využíval jsem metodu `search()` umožňující definovat kromě výchozího kontextu také vyhledávací filtry a další faktory vyhledávaní. Metoda vrací výsledky reprezentované rozhraním `NamingEnumeration` popsaným výše.

`Attributes` - Rozhraní reprezentující kolekci vrácených atributů.

`InitialDirContext` - Třída reprezentující výchozí adresářový kontext. Veškeré operace nad adresářovými službami jsou volány relativně vůči tomuto kontextu. V konstruktoru této třídy je předávána množina parametrů reprezentující prostředí kontextu, se kterým budeme pracovat. Parametry jsou zastupovány třídou `java.util.HashTable` nebo některou z jejich subtříd. V aplikaci jsem využil následující parametry:

- `Context.INITIAL_CONTEXT_FACTORY` – Specifikuje název třídy typu factory, která bude použita k vytvoření výchozího kontextu.
- `Context.SECURITY_AUTHENTICATION` – uchovává hodnotu určující způsob autentizace uživatele
- `Context.SECURITY_PRINCIPAL` – uchovává DN uživatele přistupujícího ke službě
- `Context.SECURITY_CREDENTIALS` – uchovává doplňující údaje pro autentizaci (heslo, klíč nebo certifikát)
- `Context.SECURITY_PROTOCOL` – uchovává hodnotu, specifikující bezpečnostní protokol, který bude použit
- `Context.PROVIDER_URL` – uchovává adresu poskytovatele služeb

`SearchControls` - Třída zapouzdřující faktory, které určují druh vyhledávání a co má být vráceno jako výsledek vyhledávání pomocí metod:

- `setSearchScope()` - umožňuje definovat rozsah prohledávání
- `setReturningAttributes()` - určuje seznam atributů k vrácení
- `setCountLimit()` - udává maximální počet vrácených objektů
- `setTimeLimit()` - udává maximální dobu pro zpracování výsledků

`SearchResult` - Třída reprezentující položku z NamingEnumeration. Zavoláním metody `getAttributes()` nad touto třídou získáme hodnoty vracených atributů.

Pro shrnutí opět uvedu ukázkový příklad zahrnující všechny popsané třídy a rozhraní. Pro účely porovnání práce s JDBC a JNDI jsem zvolil stejný příklad jako v ukázce práce s databází, tedy výpis načtených uživatelských jmen, uložených v atributu `cn`. Taktéž s ošetřením možných výjimek. Komunikace bude probíhat nad zabezpečeným protokolem, což vyžaduje ověření certifikátem i autorizací uživatele. K tomu je potřeba zadat, kde lze certifikát najít. Pomocí příkazu `System.setProperty` určíme uložením hodnoty do příslušné proměnné `javax.net.ssl.trustStore`, kde se nachází úložiště s daným certifikátem. V našem případě se jedná o soubor „ldap.jks“.

```
Hashtable env = new Hashtable();
String url = "ldaps://ldap.vsb.cz:636";
String[] attrIDs = ["cn"];
System.setProperty("javax.net.ssl.trustStore", "ldap.jks");
System.setProperty("javax.net.debug","ssl");
env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
env.put(Context.SECURITY_AUTHENTICATION, "simple");
env.put(Context.SECURITY_PRINCIPAL, "cn=idm_reader,ou=HOME,o=CVT");
env.put(Context.SECURITY_CREDENTIALS, "password");
env.put(Context.SECURITY_PROTOCOL, "ssl");
env.put(Context.PROVIDER_URL, url);
try {
    DirContext ctx = new InitialDirContext(env);
   SearchControls sc = new SearchControls();
   sc.setSearchScope(SearchControls.ONELEVEL_SCOPE);
   sc.setReturningAttributes(attrIDs);
   results = ctx.search("o=0", "(objectclass=person)", sc);
   Attributes attributes;
   while (results.hasMore()) {
      searchResult = (SearchResult) results.next();
       attributes = searchResult.getAttributes();
      System.out.println(attributes.get("cn").get());
   }
   ctx.close();
} catch (NameNotFoundException e) {
    System.out.println("Výchozí kontext nebyl nalezen");
} catch (NamingException e) {
    System.out.println("Chyba při práci s adresářem");
}
```

Výpis 8: Ukázka práce s databází

5.4.3 Nastavení

Aplikace implementuje veškerá požadovaná nastavení. Vše je navrženo tak, aby aplikace dokázala v co největší míře zamezit chybám. Nastavení tedy obsahuje následující omezení:

- mapování
  - k vytvoření nového mapování je samozřejmostí, aby existovaly atributy, ze kterých může být mapování vytvořeno
  - každý atribut může být mapován pouze jednou
  - mapovány mohou být pouze atributy podobného typu (nelze tedy například mapovat atributy typu `Integer` a `Date`)
- atributy
  - nelze odstranit mapovaný atribut

  - nelze odstranit atribut, který je využíván jiným atributem s kardinalitou 1:N nebo M:N

- tabulky
  - nelze odstranit výchozí tabulky uživatelů a skupin
  - nelze odstranit tabulky, ve kterých je definován alespoň jeden atribut

5.4.4 Porovnávání

Hlavní část celé aplikace. Při porovnávání záznamů uživatelů nebo skupin jsou prováděny následující hlavní úkony:

1. připojení k databázi
2. připojení k LDAP
3. načtení jedinečných klíčů záznamů z databáze
4. načtení jedinečných klíčů záznamů z LDAP
5. vzájemné porovnání načtených klíčů
6. porovnání atributů u korespondujících klíčů
7. ukončení připojení k databázi a LDAP
8. vyobrazení výsledků

Připojení k databázi a LDAP

Před samotným porovnáváním je nutno vytvořit připojení k databázi a adresářovému serveru, mezi kterými budou data porovnávána. Pro připojení k databázi je nutno zadat adresu serveru, kde se databáze nachází, uživatelské jméno a heslo pro přístup k databázi, typ databáze, její název a port, přes který je možno se k ní připojit.

U připojení k LDAP máme dvě možnosti:

1. veřejné připojení - stačí zadat adresu LDAP serveru. V tomto případě však máme možnost porovnávat pouze veřejné atributy
2. neveřejné (zabezpečené) připojení - s adresou serveru je nutno zadat ještě DN uživatele přistupujícího k adresáři a jeho heslo. Při tomto typu připojení probíhá veškerá komunikace mezi aplikací a adresářovou službou nad zabezpečeným protokolem LDAPS.

Načtení jedinečných klíčů

Po úspěšném připojení k databází a LDAP jsou načteny jedinečné hodnoty ze zvolených atributů, pomocí kterých bude zjišťováno, které záznamy se nachází pouze v databázi

nebo v LDAP, a které záznamy budou mezi sebou vzájemně porovnány. To je implementováno tak, že kolekce klíčů načtených z databáze je postupně procházena a ke každému klíči je hledán jemu odpovídající v kolekci načtené z LDAP. Při nenalezení je záznam s takovouto jedinečnou hodnotou vyhodnocen jako „nalezen pouze v databázi“, v opačném případě dojde k porovnávání. Při porovnávání takových záznamů, je hodnota z kolekce klíčů načtených z LDAP odstraněna. Záznamy se zbylými jedinečnými hodnotami v této kolekci jsou po průchodu všech hodnot z databáze vyhodnoceny jako „nalezeny pouze v LDAP“.

Vzájemné porovnávání

Po nalezení odpovídajících klíčů v obou kolekcích přichází na řadu porovnávání všech mapovaných atributů. Hodnoty těchto atributů je nutno v tuto chvíli načíst. Pro tyto účely byl před porovnáváním z mapovaných atributů vytvořen SQL příkaz pro načítání záznamů z databáze a sestaveno pole atributů, které bude vráceno při čtení z LDAP. Všechny načtené atributy jsou poté postupně porovnány podle toho, jakého jsou typu:

- numerické atributy - je testovaná rovnost hodnot
- textové atributy - porovnávání řetězců
- datumové atributy - v tomto případě je nutno převést načtené hodnoty na jeden společný datumový typ, aby byly porovnatelné. Po převedení se porovnává pouze rok, měsíc a den.
- vícehodnotové atributy - vzájemně jsou porovnány obě kolekce atributů. Obsahuje-li jedna z kolekcí záznam, který neobsahuje kolekce druhá, jsou atributy vyhodnoceny jako nesynchronizované

Veškeré hodnoty porovnávaných atributů jsou tedy načítány vždy pouze pro aktuální porovnávání dvou záznamů. Načtení všech atributů zároveň s načítáním klíčů by výrazně urychlila aplikaci, protože by odpadly časté přístupy do databáze a LDAP, na druhou stranu však při několika desítkách tisíc záznamů s přibližně desítkou atributů by aplikace pohltila velkou část paměti.

Ukončení připojení

Na závěr porovnávání je nutno korektně ukončit práci s databází a adresářovou službou.

Vyobrazení výsledků

Poté je již možno nechat si přehledně zobrazit nalezené nesynchronizované záznamy. Nyní tedy máme možnost se podívat například, ve kterých atributech se korespondující záznamy neshodují. Nabízí se také možnost níže popsaného filtrování a vyhledávání výsledků.

5.4.5 Filtrování

V aplikaci jsou implementovány dva různé, mírně odlišně způsoby filtrování. Jedná se o filtrování záznamů nalezených pouze v databázi nebo v adresáři a filtrování neidentických záznamů. V obou případech lze kombinovat více podmínek najednou a lze si také vybrat, zda musí být splněny všechny zadané podmínky současně nebo pouze jedna z nich.

V prvním případě jsou výsledky filtrovány tak, že po aplikaci filtrů jsou zobrazeny pouze záznamy vyhovující podmínkám typu

`<atribut><operátor><hodnota>`

Operátory lze volit podle typu atributů:

- numerické a datové typy atributů: je rovno, není rovno
- textové typy atributů a vícehodnotové atributy: obsahuje, neobsahuje

Například je možné si nechat zobrazit pouze studenty z fakulty elektrotechniky a informatiky. Dejme tomu, že máme definován atribut s názvem `groups`, do kterého jsou načítány skupiny, do kterých uživatel patří. Podmínka by poté vypadala následovně: `groups obsahuje STU_FEI`

V případě neidentických záznamů lze filtrovat výsledky tak, že můžou být zobrazeny pouze ty, které se liší nebo naopak neliší ve vybraných atributech.

Můžeme si tak nechat zobrazit třeba pouze záznamy, které se liší v atributech obsahující e-maily, ale neliší se v atributech s uloženými skupinami.

Veškeré filtry je možno dočasně deaktivovat a následně opět aktivovat bez nutnosti rušení filtrů a opětovného definování.

5.4.6 Vyhledávání

Součástí práce s výsledky porovnávání je také možnost prohledávání výsledků. To může být užitečné při velkém počtu záznamů ve výsledcích. Jednoduše pak zadáním jednoznačné hodnoty hledaného záznamu můžeme zjistit, zda:

- záznam je uložen pouze v databázi
- záznam je uložen pouze v LDAP
- záznam obsahuje rozdílné hodnoty v některých z mapovaných atributů
- záznam neobsahuje žádné rozdíly v hodnotách mapovaných atributů

Nachází-li se záznam v některé z kolekcí výsledků, můžeme si ihned zobrazit jeho detailní informace.

5.4.7 Uživatelské rozhraní

Uživatelské rozhraní je zpracováno pomocí komponent z knihoven `java.awt` a zejména `javax.swing`. Hlavním cílem uživatelského rozhraní bylo přehledné zobrazení porovnávání, jeho výsledků a prostředí pro konfiguraci onoho porovnávání.

Výsledky jsou zobrazovány v hlavním okně aplikace, rozděleny do dvou částí, a to na výsledky porovnávání uživatelů a porovnávání skupin. Obě tyto kategorie jsou doplněny o možnosti filtrování a vyhledávání ve výsledcích. Jsou tedy vytvořeny i dialogy pro snadné definování podmínek pro filtrování nebo vyhledávání výsledků. Pro veškeré nalezené výsledky je možno zobrazit si detailnější informace, které byly načteny z databáze či adresáře. Ukázka hlavního okna s vyobrazením nalezených výsledků po porovnávání obou kategorií je na obrázku 13 na straně 46.

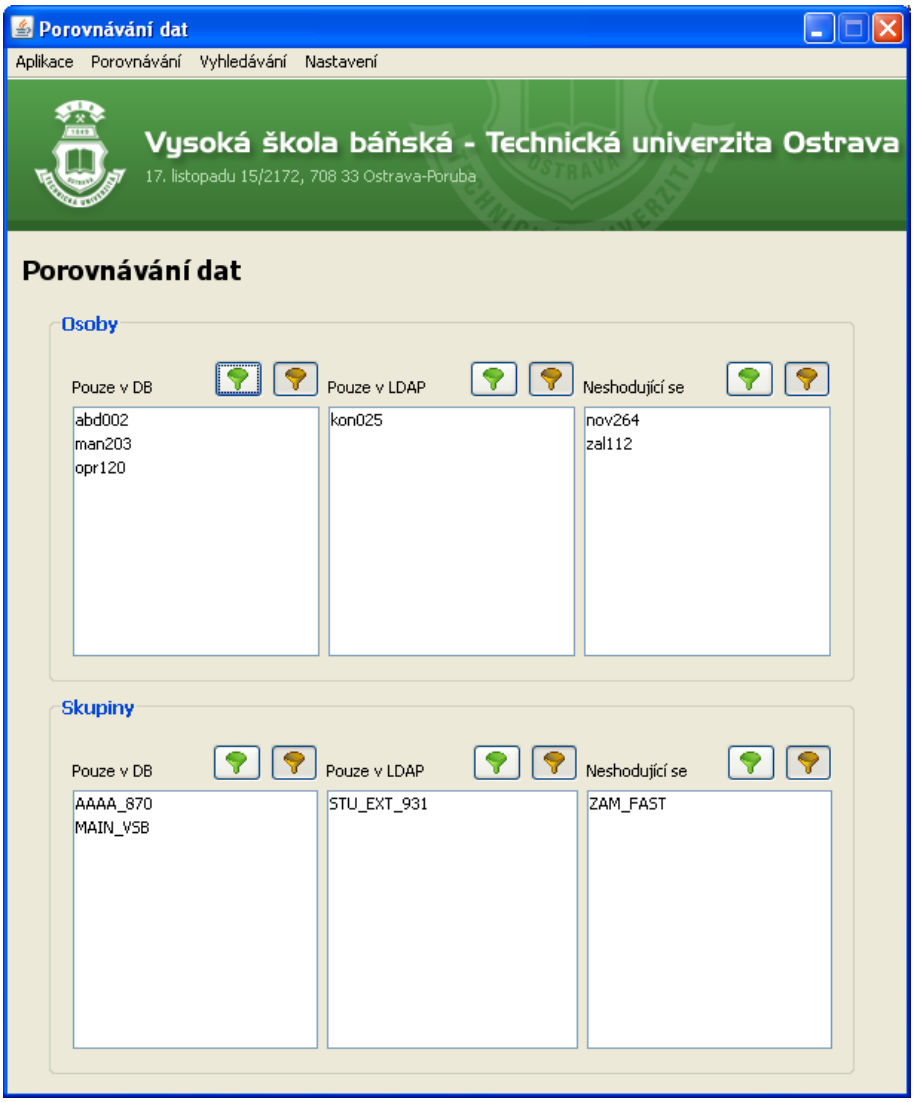
Druhou sekcí je dialog pro konfiguraci. Ten je rozdělen na pět hlavních kategorií pro definování mapování, atributů, databázových tabulek a schématu, dalšího nastavení pro potřeby porovnávání a v poslední řadě pro definování údajů nutných pro přihlášení k databázi a LDAP adresáři, které mezi sebou budeme chtít porovnávat. V první fázi bylo prostředí pro konfiguraci, zejména definování databázových atributů mírně neintuitivní, to však bylo později zpřehledněno a definování atributů se tak pro uživatele stalo snazším. Při definování veškerých atributů jsou pečlivě kontrolovány veškeré uživatelské vstupy, pro zabránění chybného chování aplikace při porovnávání. Aplikace nabízí možnost veškerá tato nastavení uložit do XML souboru. Uživatel tak může mít nadefinováno více nastavení a při různých porovnáváních pak nemusí pokaždé pracně upravovat nastavení, ale pouze stačí načíst soubor s daným nastavením. Na obrázku 14 na straně 47 je zachyceno nastavení mapování pro porovnávání uživatelů a skupin.

Třetí hlavní část grafického rozhraní je dialog pro zobrazování průběhu aktuálního porovnávání, ve kterém je uživatel informován o počtu již porovnaných záznamů, nalezených nesynchronizovaných záznamech či případně o nastalých chybách při práci s databází nebo adresářem.

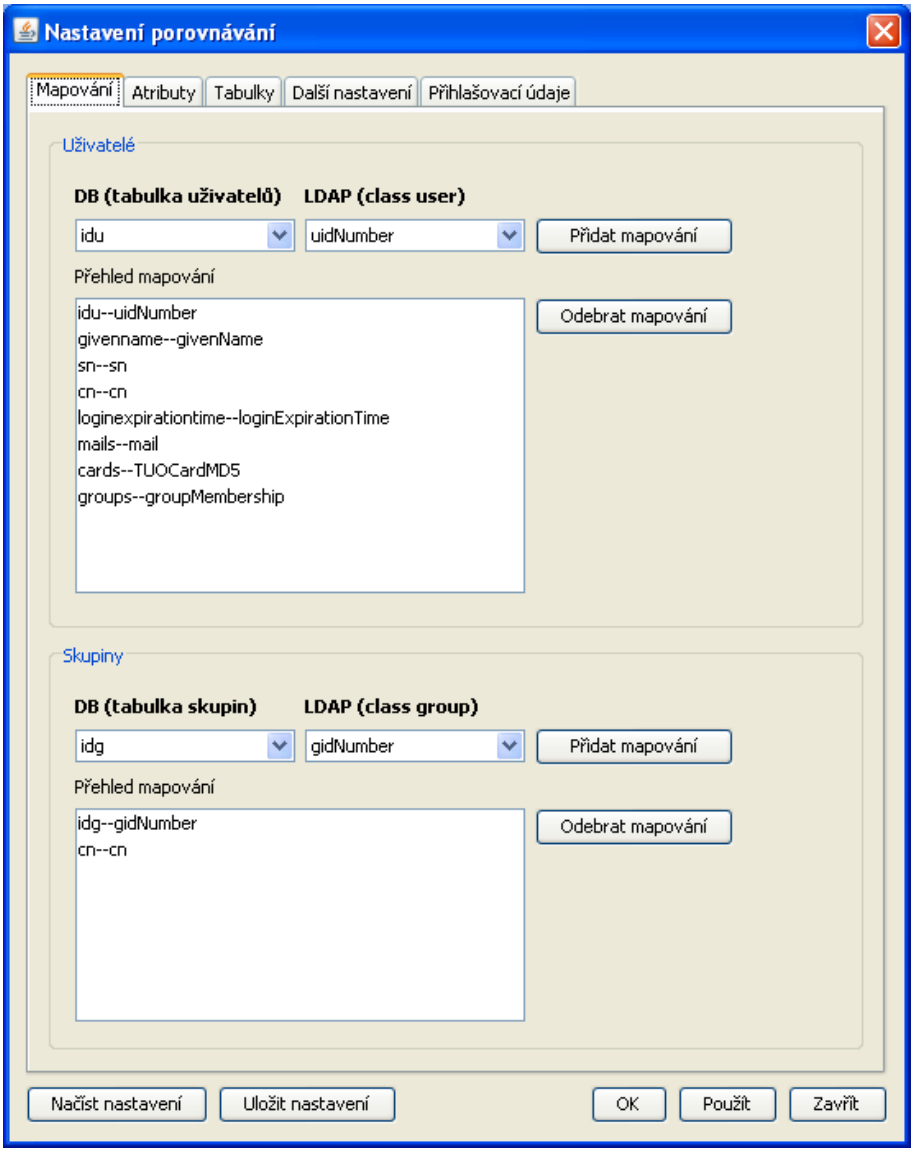
5.4.8 Vývojové prostředky

Zde jsou uvedeny prostředky, které jsem využíval při vývoji aplikace, jak pro programování aplikace, tak pro přístupy k datům z databází a LDAP adresáře, pro potřeby kontroly správnosti funkcí aplikace.

- vývojové prostředí NetBeans 5.5.1
- programovací jazyk Java
- Java Development Kit 1.6
- IBM DB2 Control Center
- JXplorer - A Java LDAP Browser



Obrázek 13: Hlavní okno uživatelského rozhraní



Obrázek 14: Dialog pro konfiguraci porovnávání

5.5 Testování

Testování aplikace probíhalo nad ostrými daty v LDAP a daty v testovací databázi, které nebyly zcela aktuální. To však bylo výhodou pro ověřování funkčnosti aplikace při porovnávání, protože se objevovaly spousty nesynchronizovaných záznamů, které dopomohly k vyladění aplikace a umožnění spuštění porovnávání nad relevantními daty, pro které byl program primárně vyvíjen.

6 Závěr

Cílem této práce bylo navrhnout a implementovat reálně použitelnou aplikaci pro porovnávání dat týkajících se uživatelů a skupin figurujících na VŠB – TU Ostrava. V souvislosti s tím jsem se blíže seznámil s problematikou relačních databází, adresářových služeb a zejména správou identit a jejím nasazováním do organizací. U těchto teoretických části byly popsány převážně základní vlastnosti, možnosti využití a jejich hlavní funkce. Kde to bylo možné, byly pro lepší pochopení daného tématu uvedeny také ukázkové příklady. Problematika těchto témat je však velice obsáhlá a v rozsahu této práce tedy nemohlo být uvedeno vše.

V další části jsem popisoval vyvíjený program. Tato sekce zahrnuje návrh aplikace a popis implementace funkcí, které aplikace nabízí. Součástí toho byl také popis technologií, kterých bylo při vývoji využito. Zadání jsem dle mého názoru splnil. Aplikaci se v budoucnu nabízí její rozšíření a to zejména o možnosti upravení nesynchronizovaných atributů či odstranění nežádoucích záznamů přímo v aplikaci, což nebylo předmětem mého zadáni.

Během vývoje jsem si prohloubil znalosti pro práci s databázemi a adresářovými službami v jazyce Java. Práce mi taktéž přinesla nové poznatky z oblasti zmiňované teoretické části a to zejména správy identit a jejich výhody při nasazování do heterogenních systémů.

David Vaštík

7 Reference

[1] Jana Šarmanová. *Teorie zpracování dat*. VŠB – TUO, 1997. Skriptum k předmětu TZD.

[2] IBM. *DB2 Triggers*. [online], c2009 [cit. 2009-04-01]. Dostupný z URL: <http://publib.boulder.ibm.com/infocenter/db2luw/v8/>.

[3] The Free Encyclopedia Wikipedia. *Lightweight Directory Access Protocol*. [online], c2009 [cit. 2009-04-01]. Dostupný z URL: <http://en.wikipedia.org/wiki/ldap/>.

[4] Jiří Sitera. *Adresářové služby – úvod do problematiky*. [online], 2000 [cit. 2009-04-01]. Dostupný z URL: <http://www.cesnet.cz/doc/techzpravy/2000-4/>.

[5] RFC 2222. *Simple Authentication and Security Layer*. [online], c1997 [cit. 2009-04-01]. Dostupný z URL: <http://www.ietf.org/rfc/rfc2222.txt>.

[6] RFC 2849. *LDAP Data Interchange Format*. [online], c2000 [cit. 2009-04-01]. Dostupný z URL: <http://www.ietf.org/rfc/rfc2849.txt>.

[7] OASIS. *Directory Services Markup Language v2.0*. [online], 2002 [cit. 2009-04-01]. Dostupný z URL: <http://www.oasis-open.org/committees/dsml/docs/DSMLv2.doc>.

[8] *Novell eDirectory 8.8 Documentation*. [online], c2009 [cit. 2009-04-01]. Dostupný z URL: <www.novell.com/documentation/edir88/index.html>.

[9] *Sun Directory Server*. [online], c2009 [cit. 2009-04-01]. Dostupný z URL: <http://www.sun.com/software/products/directory_srvr_ee/dir_srvr/index.xml>.

[10] OpenLDAP Foundation. *The OpenLDAP Project*. [online], c2009 [cit. 2009-04-01]. Dostupný z URL: <http://www.openldap.org/>.

[11] *Novell Identity Manager 3.6 Documentation*. [online], 2008 [cit. 2009-04-01]. Dostupný z URL: <http://www.novell.com/documentation/idm36/index.html>.

[12] Zdeněk Burda. *Sun Java Identity Manager*. [online], 2007 [cit. 2009-04-01]. Dostupný z URL: <http://www.abclinuxu.cz/clanky/site/sprava-uzivatelu-v-siti-2-sun-java-identity-manager>.

[13] Martin Lasoň. *Zkušenosti s nasazováním Identity Managementu na VŠB – TUO*. [online], 2007 [cit. 2009-04-01]. Dostupný z URL: <http://www.europen.cz/Proceedings/TWoDI/idm-vsb.pdf>.

[14] The Java™ Tutorials. *JDBC(TM) Database Access*. [online], c2008 [cit. 2009-04-01]. Dostupný z URL: <http://java.sun.com/docs/books/tutorial/jdbc/>.

[15] Rosanna Lee. *The JNDI Tutorial*. [online], c2004 [cit. 2009-04-01]. Dostupný z URL: <http://java.sun.com/products/jndi/tutorial/>.

A Obsah CD

Součástí této práce je i přiložené CD. Popis adresářové struktury s příslušným obsahem je uveden v následujícím přehledu.

- `/` - obsahuje český a anglický abstrakt této práce v textových souborech
- `/text` - obsahuje text této práce ve formátu PDF
- `/text/latex` - obsahuje text této práce ve zdrojovém formátu LaTeX
- `/doc` - obsahuje uživatelskou příručku ve formátu PDF
- `/javadoc` - obsahuje programátorskou dokumentaci k projektu ve formátu HTML
- `/projekt` - obsahuje kompletní projekt do prostředí NetBeans (včetně potřebných ovladačů a knihoven pro práci s databázemi)